Вычислительный центр им. А. А. Дородницына
Федерального исследовательского центра «Информатика и управление»
Российской академии наук

На правах рукописи

Прокофьев Пётр Александрович

Корректное распознавание по прецедентам: построение логических корректоров общего вида и вычислительные аспекты

Специальность 05.13.17 —
«Теоретические основы информатики»

Диссертация на соискание учёной степени
кандидата физико-математических наук

Научный руководитель:
доктор физико-математических наук
Дюкова Е. В.

Москва – 2016

Оглавление

Введение

Актуальность темы. Центральной проблемой, на которую нацелена диссертационная работа, является корректное распознавание по прецедентам. Исследуется множество объектов, которое может быть разбито на конечное число классов. О характере этого разбиения можно судить только по обучающей выборке (конечному набору прецедентов). Каждый объект может быть представлен в виде числового вектора, полученного в результате наблюдения или измерения определённых характеристик объекта. Такие характеристики называются признаками. Требуется построить алгоритм распознавания, который по предъявленному признаковому описанию объекта определяет, к какому классу следует отнести этот объект. Алгоритм распознавания, безошибочно классифицирующий прецеденты, называется корректным. Важным показателем качества корректного алгоритма распознавания является его обобщающая способность (частота ошибок на объектах, не участвующих в обучении).

В случае целочисленных признаков задача корректного распознавания достаточно эффективно решается методами *логического подхода* [1–9]. Базовым для этого подхода является понятие *элементарного классификатора* (эл.кл.) — элементарной конъюнкции, заданной на признаковых описаниях объектов. Говорят, что эл.кл. выделяет некоторый объект, если он принимает значение 1 на признаковом описании этого объекта. Традиционно при построении логических алгоритмов распознавания используются корректные эл.кл. Эл.кл. называется корректным для некоторого класса, если совокупность выделяемых им прецедентов является подмножеством либо этого класса, либо объединения остальных классов. Если все прецеденты, выделяемые корректным эл.кл., принадлежат одному классу, то такой эл.кл. называется представительным набором. Известно, что алгоритмы голосования по представительным наборам наиболее успешно применяются для задач распознавания с признаками небольшой значности (под значностью признака понимается число его допустимых

значений). В этом случае, как правило, удаётся найти достаточное количество информативных представительных наборов.

Проблемными для классических логических алгоритмов распознавания являются задачи с вещественными признаками и целочисленными признаками большой значности. Для повышения эффективности решения таких задач применяются следующие методики: 1) ищутся логические закономерности (понятие логической закономерности обобщает понятие эл.кл. на случай вещественных признаков) [10–12]; 2) вещественные признаки трактуются как целочисленные высокой значности и выполняется корректная перекодировка признаков с целью понижения их значности [13, 14]; 3) строятся корректные алгоритмы распознавания на базе произвольных, не обязательно корректных эл.кл. (*алгебро-логический подход*) [15–19].

В основе алгебро-логического синтеза распознающих алгоритмов лежат понятия и методы двух подходов: логического и алгебраического. *Алгебраический подход*, развиваемый школой Ю. И. Журавлёва [20–25], применяется, когда требуется скорректировать работу нескольких различных алгоритмов, каждый из которых безошибочно классифицирует лишь часть обучающих объектов. Цель коррекции — сделать так, чтобы ошибки одних алгоритмов были скомпенсированы другими, и качество результирующего алгоритма оказалось лучше, чем каждого из базовых алгоритмов в отдельности.

В [15] вводится понятие *корректного набора эл.кл.*, которое впоследствии становится основным для алгебро-логического подхода. Алгоритмы распознавания, основанные на голосовании по корректным наборам эл.кл., называются *логическими корректорами*. Фактически эл.кл. выступают в роли базовых распознающих алгоритмов и корректируются булевыми функциями. Основной задачей этапа обучения логических корректоров является поиск корректных наборов эл.кл. с хорошей распознающей способностью. Каждый корректный набор эл.кл. однозначно соответствует покрытию булевой матрицы, построенной специальным образом по обучающей выборке. При большой значности признаков приходится обрабатывать матрицы, размер которых экспоненциально зависит от объема обучающей информации. Поэтому возникает проблема применения логических корректоров на практике.

В [18] разработаны первые практические модели логических корректоров. В указанной работе для снижения вычислительных затрат предложено использовать эл.кл.

ранга 1 и поиск корректных наборов эл.кл. с распознающей способностью, близкой к максимальной, осуществлять генетическим алгоритмом. Установлено, что логические корректоры с монотонными корректирующими функциями (монотонные логические корректоры) имеют более высокую обобщающую способность, чем с произвольными.

Проведённые в [26] эксперименты показывают, что на прикладных задачах с большой значностью признаков монотонные логические корректоры опережают классические логические алгоритмы распознавания. В случае небольшой значности признаков ситуация обратная. По-видимому, ограничение, налагаемое на ранг эл.кл., не позволяет построить в последнем случае логические корректоры с хорошей обобщающей способностью.

Актуальной задачей является расширение границ применимости алгебро-логического подхода за счёт построения и исследования новых, более совершенных моделей логических корректоров. Перспективным направлением является использование других семейств корректирующих функций, отличных от семейства монотонных булевых функций и множества всех булевых функций. Также необходимо разработать методику обучения логических корректоров, позволяющую с небольшими вычислительными затратами получать высокое качество распознавания.

Трудности вычислительного характера, возникающие при реализации как классических логических алгоритмов распознавания, так и логических корректоров, связаны с необходимостью решать известные своей сложностью дискретные задачи. Среди этих задач главной считается *дуализация*. Это задача перечисления неприводимых покрытий булевой матрицы. Говорят, что алгоритм дуализации имеет полиномиальную задержку, если каждый его шаг (построение очередного решения) осуществляется за время, полиномиально зависящее от размера входа [27]. Вопрос о полиномиальной разрешимости дуализации поставлен более 40 лет назад, однако до сих пор ответ на этот вопрос не найден. В зарубежной литературе наибольшее распространение получил инкрементальный принцип построения алгоритмов дуализации, и в этом направлении лучшим теоретическим результатом считается построение инкрементальных алгоритмов, квазиполиномиальная сложность которых обоснована «в худшем» случае [28–31]. Однако на реальных задачах наилучшие результаты показывают так называемые асимптотически оптимальные алгоритмы дуализации, имеющие теоретическое обоснование эффективности «в среднем» [32–40].

Цель данной работы — развитие методов алгебро-логического подхода к корректному распознаванию по прецедентам, а именно построение логического корректора общего вида, позволяющего в определенной степени повысить качество распознавания и снизить вычислительные затраты этапа обучения; разработка конструкций асимптотически оптимальных алгоритмов дуализации для решения задач большого размера.

Решена следующая группа задач.

1. Обобщено понятие корректного набора эл.кл. Описана схема логического корректора общего вида. Выявлено место классических логических алгоритмов распознавания и ранее построенных логических корректоров в этой схеме.

2. Разработана и исследована более совершенная модель логического корректора с корректирующими функциями из семейства, отличного от семейства монотонных булевых функций и множества всех булевых функций.

3. Разработана методика повышения качества распознавания и скорости обучения логических корректоров. Проведено экспериментальное обоснование эффективности предложенной методики.

4. Модифицированы конструкции ряда асимптотически оптимальных алгоритмов дуализации с целью снижения времени их работы. Экспериментально показано превосходство построенных алгоритмов дуализации по сравнению с другими известными алгоритмами дуализации.

Методы исследования. Применялись методы дискретной математики, алгебры, математической логики, анализа алгоритмов и вычислительной сложности. Экспериментальное исследование проводилось с использованием программно-алгоритмического комплекса, разработанного автором.

Научная новизна. В работе строится логический корректор общего вида, для описания которого используется язык предикатов. Вводятся понятия корректного и представительного предиката. Каждый предикат однозначно определяется некоторым набором эл.кл. и корректирующей функцией этого набора.

Впервые решается важная методологическая задача обобщения логического и алгебро-логического синтеза корректных алгоритмов распознавания. Предложен-

ная в работе схема синтеза корректных алгоритмов распознавания может быть использована для описания как классических логических распознающих алгоритмов, так и ранее построенных логических корректоров.

В рамках общей схемы построена новая модель практического логического корректора POLAR, голосующего по предикатам специального вида и имеющего поляризуемую корректирующую функцию. Булева функция называется *поляризуемой*, если она по каждой переменной либо монотонно не возрастает, либо монотонно не убывает. Семейство монотонных булевых функций содержится в семействе поляризуемых булевых функций. Ранее поляризуемые функции общего вида в качестве корректирующих не использовались.

Предложена новая методика снижения вычислительных затрат и повышения качества распознавания логических корректоров. На этапе обучения логического корректора семейства голосующих предикатов формируются итеративно по принципу *бустинга* [41, 42]. Снято ограничение на ранг эл.кл., и поиск корректных наборов эл.кл. осуществляется в рамках *локальных базисов классов* — предварительно построенных корректных наборов, состоящих из информативных эл.кл. Разработаны итеративные алгоритмы формирования «хороших» локальных базисов. Отметим, что в [26] успешно реализован логический корректор, локальные базисы которого строятся стохастическим алгоритмом. Вообще говоря, идея применения локальных базисов в алгебраическом подходе впервые встречается в [43, 44].

В диссертационной работе построен ряд новых асимптотически оптимальных алгоритмов дуализации, в основе которых лежит следующий подход [35, 37]. Исходная перечислительная задача Z заменяется на более «простую» перечислительную задачу Z_1, имеющую тот же вход и решаемую с полиномиальной задержкой. При этом, во-первых, множество решений задачи Z_1 содержит множество решений задачи Z, и во-вторых, почти всегда с ростом размера входа число решений задачи Z_1 асимптотически равно числу решений задачи Z. Теоретическое обоснование данного подхода базируется на получении асимптотик для типичного числа решений каждой из задач Z и Z_1.

Таким образом, в отличие от «точного» алгоритма с полиномиальной задержкой, асимптотически оптимальному алгоритму разрешено делать «лишние» полиномиальные шаги. Лишний шаг — это построение такого решения задачи Z_1, которое

либо было найдено ранее, либо построено впервые, но не является решением задачи Z. Проверка того, является ли выполненный шаг лишним должна осуществляться за полиномиальное время от размера задачи.

Работу асимптотически оптимального алгоритма дуализации A на входной матрице L наглядно можно представить в виде обхода в глубину дерева решений $T_A(L)$. Корнем дерева $T_A(L)$ является пустой набор, остальным вершинам соответствуют наборы столбцов матрицы L. Построение висячей вершины связано либо с получением неприводимого покрытия матрицы L, либо с завершением «лишнего» шага алгоритма. Если вершина H не является висячей, то каждая её дочерняя вершина образуется добавлением к H в точности одного столбца.

Построенные в работе асимптотически оптимальные алгоритмы дуализации являются лидерами по скорости счёта. Снижение вычислительных затрат достигается за счёт сокращения общего числа вершин дерева решений. Ранее при построении асимптотически оптимальных алгоритмов основные усилия по уменьшению времени счёта направлялись на сокращение числа висячих вершин дерева решений (числа лишних шагов). При этом, как правило, усложнялся шаг алгоритма.

Теоретическая значимость. Построена общая схема алгебро-логического синтеза корректных алгоритмов распознавания, основанных на голосовании по предикатам, каждый из которых является композицией некоторого корректного набора эл.кл. и его корректирующей функции. Предложен метод построения предикатов специального вида. Исследованы свойства этих предикатов.

Получены теоретические оценки скорости сходимости бустинг-алгоритма формирования семейств голосующих предикатов. На каждой итерации ищется предикат, наилучшим образом компенсирующий ошибки ранее построенных предикатов. Качество добавляемого предиката оценивается функционалом «взвешенной» информативности. Поиск предиката с максимальной информативностью сведён к специальной задаче дискретной оптимизации, обобщающей ряд известных задач [45,46]. Решение поставленной оптимизационной задачи в общем случае представляет теоретический и практический интерес.

На значительном объеме тестовых данных, включающих разнотипные модельные и прикладные задачи, проведено сравнение новых и ранее построенных асимптотически оптимальных алгоритмов дуализации с другими известными алгоритмами.

Подобное экспериментальное обоснование асимптотически оптимального подхода до сих пор не проводилось.

Рассмотрена задача поиска ветви дерева решений $T_A(L)$, началом которой является некоторая фиксированная внутренняя вершина, а концом — висячая вершина, соответствующая решению дуализации. Доказано, что эта задача NP-полна. Данный результат объясняет, почему не увенчались успехом предпринимаемые ранее попытки избавиться от лишних шагов в асимптотически оптимальных алгоритмах дуализации, основанных на обходе в глубину дерева решений $T_A(L)$.

Практическая значимость. Разработанные распознающие алгоритмы позволяют решать широкий класс прикладных задач, в которых объекты могут быть представлены целочисленными признаковыми описаниями. К таким задачам относятся компьютерный анализ речи, распознавание изображений, медицинская диагностика и пр. Как уже отмечалось, дуализация является одной из центральных дискретных перечислительных задач. К дуализации могут быть сведены многие задачи, возникающие при логическом анализе данных, к числу которых, помимо распознавания по прецедентам, относятся кластерный анализ, построение ассоциативных правил, составление расписаний и пр. Построенные в работе алгоритмы дуализации позволяют за приемлемое время решать достаточно большие прикладные задачи.

На защиту выносятся следующие результаты.

1. Создание общей схемы синтеза логических корректоров, подходящей для описания классических логических алгоритмов распознавания и ранее построенных логических корректоров.

2. Построение практического логического корректора POLAR с поляризуемой корректирующей функцией.

3. Разработка методики повышения качества распознавания и скорости обучения логических корректоров, в основе которой лежат построение локальных базисов классов и формирование семейств голосующих предикатов по принципу бустинга.

4. Построение асимптотически оптимальных алгоритмов дуализации AO1M, AO1K, AO2M, AO2K, RUNC, RUNC-M, PUNC и экспериментальное ис-

следование границ применимости этих алгоритмов в зависимости от типа и размера входа.

Достоверность полученных результатов подтверждается доказательствами сформулированных утверждений и теорем, а также результатами экспериментов, проведённых автором.

Апробация работы. Основные положения и результаты диссертации докладывались на конференциях «Электронные библиотеки: Перспективные Методы и Технологии, Электронные коллекции (RCDL-2011)» (г. Воронеж, 2011 г.), «Математические методы распознавания образов (ММРО-15)» (г. Петрозаводск, 2011 г.), «Интеллектуализация обработки информации (ИОИ-9)» (Черногория, г. Будва, 2012 г.), «Pattern Recognition and Image Analysis: New Information Technologies (PRIA-11-2013)» (г. Самара, 2013 г.), «Математические методы распознавания образов (ММРО-16)» (г. Казань, 2013 г.), «Интеллектуализация обработки информации (ИОИ-10)» (Греция, о. Крит, 2014 г.), «Математические методы распознавания образов (ММРО-17)» (г. Светлогорск, 2015 г.), на семинаре отдела Интеллектуальных систем ВЦ РАН им. А.А. Дородницына в июле 2015 г. и на семинаре «Математические модели информационных технологий» департамента анализа данных и искусственного интеллекта НИУ ВШЭ в марте 2016 г.

Публикации. По тематике работы опубликовано 15 научных работ [16, 17, 19, 47–58], в том числе 5 статей в журналах, рекомендованных ВАК [16, 49, 54, 55, 57].

Работа состоит из введения, трёх глав, заключения и списка литературы.

В первой главе даётся обзор основных подходов к построению корректных логических алгоритмов распознавания, а именно, логического, оптимизационного, алгебраического и алгебро-логического. Обобщается понятие корректного набора эл.кл., являющееся базовым для алгебро-логического подхода. Описывается общая схема синтеза логических корректоров. Строится новый практический логический корректор POLAR с поляризуемой корректирующей функцией.

Во второй главе разрабатывается методика повышения скорости обучения и качества распознавания логических корректоров. Семейства голосующих предикатов строятся итеративно по принципу *бустинга*. Поиск голосующих предикатов осуществляется в рамках *локальных базисов классов* — предварительно формируемых

корректных наборов, состоящих из информативных эл.кл. Эффективность предложенной методики тестируется на реальных данных.

В третьей главе рассматривается одна из центральных дискретных перечислительных задач — дуализация. Даётся обзор основных подходов к её решению, среди которых выделяется подход к построению асимптотически оптимальных алгоритмов. Алгоритмы, построенные в рамках этого подхода, классифицируются на два типа. Строятся новые асимптотически оптимальные алгоритмы первого типа AO1K, AO1M, AO2K и AO2M, и второго типа RUNC, RUNC-M и PUNC. Новые и ранее построенные асимптотически оптимальные алгоритмы дуализации экспериментально исследуются на большом объеме разнотипных данных.

ГЛАВА 1

Корректное распознавание по прецедентам

В данной главе даётся обзор основных подходов к построению корректных логических алгоритмов распознавания, а именно, логического, оптимизационного, алгебраического и алгебро-логического. Обобщается понятие корректного набора эл.кл., являющееся базовым для алгебро-логического подхода. Описывается общая схема синтеза логических корректоров. Строится новый практический логический корректор POLAR с поляризуемой корректирующей функцией.

1.1. Основные подходы к решению задачи корректного распознавания по прецедентам

Рассматривается задача распознавания по прецедентам с множеством объектов M, представимым в виде объединения непересекающихся подмножеств $K_1, ..., K_l$, называемых классами. Задана система целочисленных признаков $\{x_1, \ldots, x_n\}$, и каждый объект S из M описывается вектором значений признаков $(x_1(S), \ldots, x_n(S))$. Задано множество объектов $T = \{S_1, \ldots, S_m\}$ из M, и для каждого объекта $S_i \in T$ известен номер класса $y_i \in \{1, \ldots, l\}$, которому принадлежит S_i. Объекты из T называются *прецедентами* или *обучающими объектами*. Требуется по обучающей выборке T построить алгоритм $A_T : M \to \{0, 1, \ldots, l\}$, ставящий в соответствие каждому объекту S из M номер класса, которому принадлежит S, или принимающее значение 0 в случае отказа от распознавания. Указанный алгоритм называется *алгоритмом распознавания*. Множество $Y = \{0, 1, \ldots, l\}$ называется *пространством ответов* алгоритма распознавания.

Пусть задана контрольная выборка объектов $T' = \{S'_1, \ldots, S'_h\}$. Для каждого контрольного объекта S'_i известен номер класса $y'_i \in \{1, \ldots, l\}$, которому принадле-

жит S'_i. В качестве функционала качества распознающего алгоритма A_T рассмотрим функционал

$$Q(A_T, T') = \frac{1}{h} \sum_{i=1}^{h} [y'_i \neq A_T(S'_i)],$$

равный среднему числу ошибок A_T на контрольной выборке T' (здесь и далее через $[p]$ обозначается предикат, принимающий значение 1 в случае, когда выражение p истинно, и 0 — в противном случае).

При $Q(A_T, T) = 0$, алгоритм A_T называется *корректным*. Если есть уверенность, что прецедентная информация содержит мало ошибок и достаточно представительна, то требование корректности алгоритма распознавания является обоснованным. Достичь выполнения этого требования можно довольно простым способом. Например, корректным является распознающий алгоритм

$$A_T^0(S) = \sum_{S_i \in T} y_i [x_1(S) = x_1(S_i) \wedge \ldots \wedge x_n(S) = x_n(S_i)]$$

Однако алгоритм A_T^0 на практике бесполезен, поскольку он не может распознать ни один неизвестный ему объект. В случае, когда алгоритм распознавания на объектах, не участвующих в обучении, ошибается значительно чаще, чем на прецедентах, говорят об эффекте *переобучения*. При решении практических задач с этим явлением приходится сталкиваться очень часто. Качество работы алгоритма распознавания, обученного по объектам из выборки T, на объектах, не входящих в T, характеризует его *обобщающую способность*.

Пусть обучающая выборка T и контрольная выборка T' получаются случайно и независимо из одного и того же распределения вероятностей, заданного на множестве объектов M. Алгоритм распознавания считается *состоятельным*, если при заданных достаточно малых значениях точности $\varepsilon > 0$ и надёжности $\eta > 0$ вероятность выполнения неравенства $Q(A_T, T') > \varepsilon$ меньше η.

На практике состоятельность алгоритма распознавания проверяется путём вычисления эмпирических оценок, связанных с указанной вероятностью. Наиболее часто используется оценка $t \times q$*-кратного скользящего контроля*. Строятся t различных разбиений выборки T на q непересекающихся подвыборок примерно одинаковой мощности, $T = T'_{1u} \cup \ldots \cup T'_{qu}$, $u \in \{1, \ldots, t\}$. Подвыборка T'_{ku}, $u \in \{1, \ldots, t\}, k \in \{1, \ldots, q\}$, является контрольной при оценке качества распозна-

ющего алгоритма, обученного по подвыборке $T_{ku} = T \setminus T'_{ku}$. При этом распознающий алгоритм $A_{T_{ku}}$, корректный для подвыборки T_{ku}, может ошибаться на объектах из подвыборки T'_{ku}. Оценка скользящего контроля вычисляется по формуле

$$\frac{1}{tq}\sum_{u=1}^{t}\sum_{k=1}^{q} Q(A_{T_{ku}}, T'_{ku}).$$

1.1.1. Логический подход

Традиционно задача корректного распознавания в случае целочисленных признаков решается в рамках логического подхода, базовым понятием которого является понятие элементарного классификатора [3–9].

Элементарным классификатором (эл.кл.) ранга r, $r \in \{1, \ldots, n\}$, называется пара (H, σ), где $H = (x_{j_1}, \ldots, x_{j_r})$ — набор различных признаков и $\sigma = (\sigma_1, \ldots, \sigma_r)$ — целочисленный вектор, в котором σ_q — допустимое значение признака x_{j_q}, $q \in \{1, \ldots, r\}$.

Обозначим через $H(S)$, $S \in M$, целочисленный вектор $(x_{j_1}(S), \ldots, x_{j_r}(S))$. Будем говорить, что эл.кл. (H, σ) выделяет объект S (является признаковым подописанием объекта S), если $H(S) = \sigma$.

Эл.кл. (H, σ) называется корректным для класса $K, K \in \{K_1, \ldots, K_l\}$, если не существует двух прецедентов S_i и S_t, выделяемых эл.кл. (H, σ), таких, что $S_i \in K$, $S_t \notin K$. Другими словами, множество прецедентов, подописанием которых является корректный для класса K эл.кл. (H, σ), либо содержится в K, либо содержится в $\overline{K}$ (здесь и далее через $\overline{K}$, $K \subseteq M$, обозначается множество $M \setminus K$).

Корректный для класса K эл.кл. (H, σ), $H = (x_{j_1}, \ldots, x_{j_r})$, $\sigma = (\sigma_1, \ldots, \sigma_r)$, называется *тупиковым*, если для любого $q \in \{1, \ldots, r\}$ эл.кл. (H', σ'), $H' = (x_{j_1}, \ldots, x_{j_{q-1}}, x_{j_{q+1}}, \ldots, x_{j_r})$, $\sigma' = (\sigma_1, \ldots, \sigma_{q-1}, \sigma_{q+1}, \ldots, \sigma_r)$, не является корректным для K.

Обозначим через $\nu_K(H, \sigma)$ число прецедентов из K, выделяемых эл.кл. (H, σ). В зависимости от значений $\nu_K(H, \sigma)$ и $\nu_{\overline{K}}(H, \sigma)$ различают три типа корректных эл.кл.:

— корректный для K эл.кл. (H, σ) называется *представительным набором класса* K, если $\nu_K(H, \sigma) \neq 0$;

— корректный для K эл.кл. (H, σ) называется *покрытием класса* K, если $\nu_K(H, \sigma)=0$;

— корректный для K эл.кл. (H, σ) называется *антипредставительным набором класса* K, если $\nu_{\overline{K}}(H, \sigma) \neq 0$ (очевидно, антипредставительный набор класса K является покрытием класса K).

Опишем схему алгоритма голосования по корректным эл.кл. На этапе обучения для каждого класса K строятся семейства C_K и $C_{\overline{K}}$. В семейство C_K входят представительные наборы класса K, а в семейство $C_{\overline{K}}$ — покрытия класса K. Одно из семейств C_K или $C_{\overline{K}}$ может быть пустым.

Каждому эл.кл. (H, σ) приписывается положительный вес $\alpha_{(H,\sigma)}$. В простейшем случае вес представительного набора (H, σ) из C_K пропорционален значению $\nu_K(H, \sigma)$, вес антипредставительного набора (H, σ) из $C_{\overline{K}}$ пропорционален значению $\nu_{\overline{K}}(H, \sigma)$, вес покрытия класса K, не являющегося антипредставительным набором класса K, обратно пропорционален мощности семейства $C_{\overline{K}}$.

Распознавание объекта S осуществляется путём взвешенного голосования по эл.кл. построенных на этапе обучения семейств. Для каждого класса K вычисляются оценки принадлежности объекта S классу K, имеющие вид

$$\Gamma(S, K) = \sum_{(H,\sigma)\in C_K} \alpha_{(H,\sigma)}[H(S) = \sigma] + \sum_{(H,\sigma)\in C_{\overline{K}}} \alpha_{(H,\sigma)}[H(S) \neq \sigma]. \qquad (1.1)$$

Объект S относится к тому классу K, для которого оценка $\Gamma(S, K)$ имеет наибольшее значение. Если таких классов несколько, то алгоритм отказывается от распознавания и возвращает 0. Корректность алгоритма распознавания обеспечивается за счёт корректности каждого эл.кл., участвующего в голосовании.

Основная задача этапа обучения — поиск информативных корректных эл.кл. В простейшем случае информативность корректного эл.кл. (H, σ) оценивается числом, выделяемых эл.кл. (H, σ) прецедентов. Практика показывает, что информативные эл.кл., как правило, имеют небольшой ранг. Поэтому семейства C_K и $C_{\overline{K}}$ строят из тупиковых корректных эл.кл. или ограничивают допустимый ранг эл.кл.

Другим классическим логическим распознающим алгоритмом является алгоритм голосования по тестам (*тестовый алгоритм*) [1]. Набор признаков H называется *тестом*, если для любого класса K и любого прецедента $S_i \in K$ эл.кл. $(H, H(S_i))$

является представительным набором для K. Алгоритм голосования по тестам на этапе обучения строит семейство тестов $\mathcal{H}$. При распознавании объекта S для каждого класса K вычисляется оценка принадлежности S к K, в простейшем случае имеющая вид

$$\Gamma(S,K) = \frac{1}{|\mathcal{H}|}\sum_{H\in\mathcal{H}}\frac{1}{|T\cap K|}\sum_{S_i\in K}[H(S_i)=H(S)]. \tag{1.2}$$

Сопоставляя формулы (1.1) и (1.2), можно заметить, что голосование по семейству тестов $\mathcal{H}$ может быть заменено на голосование по представительным наборам, построенным по тестам из $\mathcal{H}$ и обучающим объектам. Отметим, что модель тестовых алгоритмом включается в модель *алгоритмов вычисления оценок (АВО)* [2, 59].

В случае, когда признаки имеют большую значность, большинство тупиковых эл.кл. имеют большой ранг и выделяют мало прецедентов. Такие задачи являются сложными для алгоритмов голосования по корректным эл.кл. Несмотря на то, что каждый используемый эл.кл. корректен для некоторого класса K, он плохо характеризует этот класс в целом. Обычно в таких случаях либо выполняют корректную перекодировку значений признаков с целью понижения их значности [13, 14], либо строят распознающие алгоритмы на базе произвольных, не обязательно корректных, эл.кл.

Пример 1.1. Рассмотрим «игрушечную» задачу распознавания со следующими параметрами: $m = 6$, $n = 4$, $l = 2$, признаки x_1, x_2, x_3, x_4 принимают значения из множества $\{0,1,2\}$ (множество M состоит из 81 объекта), прецедентами класса K_1 являются $S_1 = (0,1,1,0), S_2 = (1,2,0,1), S_3 = (0,1,0,1)$ и прецедентами класса K_2 являются $S_4 = (1,2,1,0), S_5 = (1,1,0,1), S_6 = (1,1,1,2)$.

Построим для этой задачи алгоритм голосования по тупиковым представительным наборам. Для наглядности каждый эл.кл. будем задавать элементарной конъюнкцией. Класс K_1 имеет 4 тупиковых представительных набора $[x_1 = 0]$, $[x_2 = 2 \wedge x_3 = 0]$, $[x_2 = 1 \wedge x_4 = 0]$, $[x_2 = 2 \wedge x_4 = 1]$, класс K_2 имеет 6 тупиковых представительных наборов $[x_4 = 2]$, $[x_1 = 1 \wedge x_2 = 1]$, $[x_1 = 1 \wedge x_3 = 1]$, $[x_1 = 1 \wedge x_4 = 0]$, $[x_2 = 2 \wedge x_3 = 1]$, $[x_2 = 2 \wedge x_4 = 0]$.

В семействе эл.кл. класса K_1 признак x_2 входит в три эл.кл., а признак x_4 — в два эл.кл. Частое использование одних и тех же признаков в разных эл.кл. связано с необходимостью обеспечить корректность каждого эл.кл. Классический алгоритм

голосования по тупиковым представительным наборам отказывается распознавать 11 объектов из 81. ■

Пример 1.2. Построим для задачи из примера 1.1 другие семейства эл.кл., $C_{K_1} = \{[x_4 = 1], \quad [x_1 = 0]\}$, $C_{K_2} = \{[x_1 = 1], \quad [x_4 = 2], \quad [x_2 = 1]\}$. Вес эл.кл. $(H, \sigma) \in C_K$, $K \in \{K_1, K_2\}$, положим равным $|C_K|^{-1}$. Эл.кл. $[x_4 = 1]$, $[x_1 = 1]$ и $[x_2 = 1]$ не являются корректными. Однако эл.кл. $[x_4 = 1]$ чаще выделяет обучающие объекты из класса K_1, а эл.кл. $[x_1 = 1]$ и $[x_2 = 1]$ — обучающие объекты из класса K_2. Благодаря этому, алгоритм голосования по эл.кл. семейств C_{K_1} и C_{K_2} является корректным. Построенный алгоритм отказывается распознавать 6 объектов из 81. ■

В примере 1.1 алгоритм голосования по тупиковым представительным наборам строит достаточно громоздкие семейства эл.кл. В примере 1.2 семейства голосующих эл.кл. состоят как из корректных, так и из некорректных эл.кл. ранга 1. Эти семейства более лаконично описывают классы, чем семейства тупиковых представительных наборов. Фактически, построен корректный распознающий алгоритм на базе произвольных, не обязательно корректных эл.кл. Регулярное построение таких алгоритмов позволяют осуществлять методы описанных ниже оптимизационного, алгебраического и алгебро-логического подходов.

Следует отметить, что в работах зарубежных авторов вводится аналогичное понятию эл.кл. понятие *emerging pattern* [60–62]. В указанных работах на базе emerging pattern построены модели распознающих алгоритмов, схема работы которых имеют мало принципиальных отличий от логических алгоритмов распознавания, разработанных отечественными авторами.

1.1.2. Оптимизационный подход

Оптимизационный подход к корректному распознаванию предполагает следующую последовательность действий. Выбирается *эвристическая информационная модель* $\mathfrak{M}_*$ — семейство допустимых распознающих алгоритмов. Вводится функционал качества распознавания прецедентов алгоритмами из $\mathfrak{M}_*$. Затем подходящим методом решается задача оптимизации этого функционала на $\mathfrak{M}_*$. Примерами применения оптимизационного подхода являются алгоритмы вычисления оценок и голосо-

вание по логическим закономерностям в задачах с вещественнозначной информацией [2, 10–12, 63].

Проиллюстрируем оптимизационный подход применительно к задаче построения корректного алгоритма голосования по некорректным эл.кл. Пусть для каждого класса K зафиксировано семейство C_K эл.кл., не обязательно являющихся корректными. Рассмотрим эвристическую информационную модель $\mathfrak{M}_*$, в которой каждый алгоритм голосует по эл.кл. из семейств $C_{K_1}, \ldots, C_{K_l}$. Различные алгоритмы из $\mathfrak{M}_*$ различаются весами, с которыми эл.кл. семейств используются в голосовании.

Пусть для распознаваемого объекта S из M вычислены оценки $\Gamma(S, K_1), \ldots, \Gamma(S, K_l)$ по формуле (1.1). Величина

$$\Delta(S, K_y) = \Gamma(S, K_y) - \max_{z \neq y} \Gamma(S, K_z)$$

называется *отступом* распознающего алгоритма на объекте S относительно класса $K_y, y \in \{1, \ldots, l\}$. Нетрудно видеть, что номер класса для распознаваемого объекта S можно найти по формуле

$$A(S) = \sum_{y=1}^{l} y[\Delta(S, K_y) > 0]. \tag{1.3}$$

Для корректности алгоритма (1.3) требуется одновременное выполнение следующих неравенств

$$\Delta(S_i, K_y) > 0, \quad \forall y \in \{1, \ldots, l\}, \quad \forall S_i \in K_y. \tag{1.4}$$

Веса $\alpha_{(H,\sigma)}, (H, \sigma) \in C_K, K \in \{K_1, \ldots, K_l\}$, можно интерпретировать как неизвестные системы неравенств (1.4). В случае, когда система (1.4) совместна, ставится задача найти такое её решение, что

$$\sum_{y=1}^{l} \sum_{S_i \in K_y} \Delta(S_i, K_y) \to \max. \tag{1.5}$$

В противном случае ставится задача найти решение её максимальной совместимой подсистемы. То есть найти веса эл.кл., для которых

$$\sum_{y=1}^{l} \sum_{S_i \in K_y} [\Delta(S_i, K_y) > 0] \to \max. \tag{1.6}$$

При использовании оптимизационного подхода можно столкнуться со следующими проблемами. Во-первых, система (1.4) может оказаться несовместной, и тогда алгоритм (1.3), построенный с найденными весами эл.кл., будет некорректным. Причиной этому может быть плохо выбранная эвристическая информационная модель $\mathfrak{M}_*$. Во-вторых, задачи (1.5) и (1.6) могут оказаться достаточно сложными для применяемого метода оптимизации, который может и не найти оптимальное решение, даже если оно существует в $\mathfrak{M}_*$.

1.1.3. Алгебраический подход

Избежать решения сложных оптимизационных задач зачастую позволяет алгебраический подход [20–25]. Требуется скорректировать работу нескольких различных алгоритмов, каждый из которых безошибочно классифицирует лишь часть обучающих объектов. Цель коррекции — сделать так, чтобы ошибки одних алгоритмов были скомпенсированы другими, и качество композиции оказалось лучше, чем каждого из базовых алгоритмов в отдельности.

Наряду с пространством ответов Y вводится *пространство оценок* E, на котором удобно задавать алгебраические операции, позволяющие строить корректирующие функции. *Корректирующей* называется функция, аргументы и значения которой лежат в E. Отображение из множества объектов M в пространство оценок E называется *алгоритмическим оператором*. Отображение оценок из E в пространство ответов Y называется *решающим правилом*. Семейства всевозможных алгоритмических операторов, корректирующих функций и решающих правил обозначим соответственно через

$$\mathfrak{M}_*^0 = \{a : M \to E\}, \qquad \mathfrak{F}_* = \bigcup_{p=0}^{\infty}\{F : E^p \to E\}, \qquad \mathfrak{M}_*^1 = \bigcup_{p=0}^{\infty}\{C : E^p \to Y\}.$$

Алгебраический подход подразумевает выполнение следующих шагов.

1. Выбирается семейство базовых алгоритмических операторов $\mathfrak{M}^0 \subseteq \mathfrak{M}_*^0$.
2. Выбирается семейство корректирующих функций $\mathfrak{F} \subseteq \mathfrak{F}_*$. Заметим, что выбранные семейства $\mathfrak{M}^0$ и $\mathfrak{F}$ порождают семейство алгоритмических операторов вида $F(b_1(S), \ldots, b_r(S))$, где $F \in \mathfrak{F}$ и $b_1, \ldots, b_r \in \mathfrak{M}^0$.
3. Выбирается семейство решающих правил $\mathfrak{M}^1 \subseteq \mathfrak{M}_*^1$.

4. Строится корректный распознающий алгоритм в виде композиции

$$A(S) = C(F_1(b_1^1(S), \ldots, b_{d_1}^1(S)), \ldots, F_r(b_1^r(S), \ldots, b_{d_r}^r(S))),$$

где $C \in \mathfrak{M}^1$, $F_t \in \mathfrak{F}$, $t \in \{1, \ldots, r\}$, и $b_q^t \in \mathfrak{M}^0$, $q \in \{1, \ldots, d_t\}$, $t \in \{1, \ldots, r\}$.

В случае двух классов ($l = 2$) в качестве пространства оценок E чаще всего берут действительную прямую $\mathbb{R}$ и используют пороговое решающее правило

$$C(e) = \begin{cases} 0, \ e = c_0, \\ 1, \ e > c_0, \\ 2, \ e < c_0, \end{cases} \qquad c_0 = \text{const}\,.$$

Если же число классов $l > 2$, то полагают $E = \mathbb{R}^l$ и используют правило голосования

$$C(e_1, \ldots, e_l) = \begin{cases} y^*, & \underset{y \in \{1,\ldots,l\}}{\operatorname{Arg\,max}} e_y = \{y^*\}, \\ 0, & \text{иначе.} \end{cases}$$

Как уже отмечалось, выбор пространства оценок тесно связан с выбором семейства корректирующих функций $\mathfrak{F}$. Как правило, для коррекции используются не все функции вида $F : E^d \to E$, а только функции, обладающие определённым свойством. В случае $E = \mathbb{R}$ примерами являются семейство линейных функций, семейство полиномов ограниченной степени, семейство монотонных функций.

Вообще говоря, в качестве базовых алгоритмических операторов могут быть использованы произвольные алгоритмы распознавания, результатом применения которых к распознаваемому объекту является некоторая оценка из E. Типичными примерами являются алгоритмы вычисления оценок, машины опорных векторов, деревья решений. Примером в некотором смысле «простейшего» базового алгоритмического оператора служит эл.кл. Для коррекции эл.кл. удобно использовать булевы функции. Далее рассматривается подход, объединяющий идеи логического и алгебраического подходов.

1.1.4. Алгебро-логический подход

Идея алгебро-логического синтеза корректных логических алгоритмов распознавания предложена в [15]. В указанной работе, введено понятие корректного набора

эл.кл., и показано, что задача построения корректного набора эл.кл. сводится к поиску покрытия булевой матрицы, специальным образом построенной по обучающей выборке. Подход развит в работах [16–19, 58], в которых рассмотрены вопросы практического применения различных моделей корректных алгоритмов распознавания, основанных на голосовании по корректным наборам эл.кл.

Пусть имеется упорядоченный набор эл.кл. $U = \big((H_1, \sigma_1), \ldots, (H_d, \sigma_d)\big)$. Набор U ставит в соответствие объекту S из M бинарный вектор $U(S)=\big([H_1(S)=\sigma_1], \ldots, [H_d(S)=\sigma_d]\big)$, который называется *откликом* набора эл.кл. U на объекте S.

Набор эл.кл. U называется *корректным* для класса K, если для любых двух обучающих объектов S_i и S_t таких, что $S_i \in K$ и $S_t \notin K$, отклики $U(S_i)$ и $U(S_t)$ различны. Другими словами, корректный для K набор эл.кл. U различает любые два обучающих объекта, один из которых принадлежит K, а другой не принадлежит. Булева функция $F(t_1, \ldots, t_d)$ такая, что $F(U(S_i)) \neq F(U(S_t))$, $S_i \in K$, $S_t \notin K$, называется корректирующей для набора U.

В [18] построен алгоритм голосования по корректным наборам эл.кл., названный логическим корректором. При обучении для каждого класса K строится некоторое непустое семейство W_K корректных наборов эл.кл. класса K. При распознавании объекта S для каждого класса K вычисляется оценка $\Gamma(S, K)$ принадлежности объекта S классу K, имеющая вид

$$\Gamma(S, K) = \frac{1}{|W_K|} \sum_{U \in W_K} \frac{1}{|T \cap K|} \sum_{S_i \in K} [U(S_i) = U(S)]. \tag{1.7}$$

Корректный для K набор эл.кл. U называется *тупиковым*, если набор эл.кл. $U' = \big((H_1, \sigma_1), \ldots, (H_{u-1}, \sigma_{u-1}), (H_{u+1}, \sigma_{u+1}), \ldots, (H_d, \sigma_d)\big)$ не является корректным для K при любом $u \in \{1, \ldots, d\}$. В [15, 18] используются тупиковые корректные наборы эл.кл.

В [15] также введено понятие монотонного корректного для класса K набора эл.кл., которое может быть сформулировано следующим образом. Набор эл.кл. $U=((H_1, \sigma_1), \ldots, (H_d, \sigma_d))$ называется *монотонным корректным для* класса K, если для любых обучающих объектов S_i и S_t таких, что $S_i \in K$ и $S_t \notin K$, существует $j \in \{1, \ldots, d\}$, для которого $H_j(S_i) = \sigma_j$ и $H_j(S_t) \neq \sigma_j$. Для монотонного корректного набора эл.кл. существует монотонная корректирующая функция $F(t_1, \ldots, t_d)$ такая, что $F(U(S_i)) = 1, S_i \in K$, и $F(U(S_t)) = 0, S_t \notin K$.

В [18] построена модель голосования по монотонным корректным наборам эл.кл., названная в работе [19] корректором МОН. Опишем схему обучения корректора МОН. При обучении для каждого класса K строится некоторое семейство W_K монотонных корректных наборов эл.кл. класса K. При распознавании объекта S для каждого класса K вычисляется оценка $\Gamma(S,K)$ принадлежности объекта S классу K, имеющая вид

$$\Gamma(S,K) = \frac{1}{|W_K|}\sum_{U\in W_K}\frac{1}{|T\cap K|}\sum_{S_i\in K}[U(S_i)\preccurlyeq U(S)]. \tag{1.8}$$

(здесь и далее для векторов $\alpha=(\alpha_1,\ldots,\alpha_d)$ и $\beta=(\beta_1,\ldots,\beta_d)$ через $\alpha\preccurlyeq\beta$ обозначается отношение $[\alpha\preccurlyeq\beta]\equiv[\alpha_p\leqslant\beta_p,\ \forall p\in\{1,\ldots,d\}]$).

В [19] введено понятие антимонотонного корректного для класса K набора эл.кл., и построен так называемый корректор АМОН. Набор эл.кл. U называется *антимонотонным корректным для класса* K, если U является монотонным для $\overline{K}$ (здесь множество $\overline{K}$ интерпретируется как класс в задаче распознавания с двумя классами K и $\overline{K}$). Для антимонотонного корректного набора эл.кл. существует монотонная корректирующая функция $F(t_1,\ldots,t_d)$ такая, что $F(U(S_i))=0, S_i\in K$, и $F(U(S_t))=1, S_t\notin K$. При обучении для каждого класса K строится некоторое семейство W_K антимонотонных корректных для K наборов эл.кл. При распознавании объекта S для каждого класса K вычисляется оценка $\Gamma(S,K)$ принадлежности объекта S классу K, имеющая вид

$$\Gamma(S,K) = \frac{1}{|W_K|}\sum_{U\in W_K}\frac{1}{|T\setminus K|}\sum_{S_i\notin K}[U(S_i)\not\preccurlyeq U(S)]. \tag{1.9}$$

Следует отметить, что в работах зарубежных авторов развивается подход Logical Analysis of Data (LAD), схожий с алгебро-логическим подходом [64–66]. Основной задачей LAD является *бинаризация* прецедентной информации, в результате которой бинарные описания «положительных» и «отрицательных» прецедентов можно различить с помощью булевой функции определенного вида, например, монотонной, пороговой функцией или булевой функцией, задаваемой элементарной дизъюнкцией. При этом результат бинаризации должен быть оптимальным, в том смысле, что бинарные описания объектов должны быть, как можно короче. Для решения этой задачи применяются методы алгебры логики, дискретной математики, линейного и целочисленного программирования.

1.2. Общая схема построения логического корректора

Определяются понятия корректного и представительного предиката класса. С использованием этих понятий описывается общая схема алгебро-логического синтеза корректных алгоритмов распознавания. В рамках предложенной схемы рассматриваются классические логические распознающие алгоритмы и ранее построенные логические корректоры.

1.2.1. Понятие корректного предиката как обобщение понятия корректного набора элементарных классификаторов

Из соображения удобства перейдём на язык предикатов, заданных на множестве объектов M. Будем говорить, что предикат B *корректен для класса* K, если множество прецедентов, на которых предикат B равен 1, является подмножеством либо $T \cap K$, либо $T \setminus K$. Корректный для класса K предикат B будем называть *представительным для класса* K, если существует прецедент $S_i \in K$ такой, что $B(S_i) = 1$.

Пусть $U = ((H_1, \sigma_1), \ldots, (H_d, \sigma_d))$ — набор эл.кл. и $F(t_1, \ldots, t_d)$ — булева функция от d переменных. Обозначим через $F(U)$ предикат, задаваемый композицией

$$F(U(S)) = F([H_1(S) = \sigma_1], \ldots, [H_d(S) = \sigma_d]),\ S \in M.$$

Обобщим понятие корректного набора эл.кл. и введём понятие представительного набора эл.кл. Набор эл.кл. U будем называть *корректным (представительным)* для класса K, если существует булева функция F такая, что предикат $F(U)$ является корректным (представительным) для K. Функция F называется *корректирующей* для набора эл.кл. U относительно класса K.

1.2.2. Алгоритм голосования по корректным предикатам

Построим логический корректор общего вида, основанные на голосовании по корректным предикатам.

На этапе обучения для каждого класса K строятся два семейства Z_K и $Z_{\overline{K}}$ предикатов на множестве объектов M. Каждый предикат семейства Z_K является представительным для класса K. Каждый предикат семейства $Z_{\overline{K}}$ является кор-

ректным, но не является представительным для K. Предикату B приписывается вес $\alpha_B > 0$.

Распознавание осуществляется взвешенным голосованием по корректным предикатам, построенным на этапе обучения. При распознавании объекта S для каждого класса K вычисляется оценка $\Gamma(S, K)$ принадлежности объекта S классу K, имеющая вид

$$\Gamma(S, K) = \sum_{B \in Z_K} \alpha_B B(S) - \sum_{B \in Z_{\overline{K}}} \alpha_B B(S).$$

Описанный распознающий алгоритм будем называть *логическим корректором общего вида*. Для обеспечения его корректности достаточно, чтобы было справедливо

Утверждение 1.1. *Пусть A — логический корректор общего вида и $Z_{K_1}, \ldots, Z_{K_l}$, $Z_{\overline{K}_1}, \ldots, Z_{\overline{K}_l}$ — семейства предикатов, по которым осуществляется голосование при распознавании объектов. Алгоритм A корректен, если для любого класса K и любого прецедента $S_i \in K$ выполняется одно из двух условий:*

1) в семействе Z_K найдётся предикат, выделяющий S_i;

2) для каждого класса K' такого, что $K' \neq K$, в семействе $Z_{\overline{K'}}$ найдётся предикат, выделяющий S_i.

Доказательство. Пусть Z — семейство предикатов и S — объект из M. Обозначим $b(Z, S) = \{B \in Z : B(S) = 1\}$. Зафиксируем класс K и объект $S_i \in K$.

1) Если $b(Z_K, S_i) \neq \varnothing$, то $\Gamma(S_i, K) > 0$. Поскольку $\forall K' \neq K$, $\Gamma(S_i, K') \leqslant 0$, отступ $\Delta(S_i, K) > 0$, и объект S_i распознаётся алгоритмом A правильно.

2) Если $b(Z_{\overline{K'}}, S_i) \neq \varnothing$, $\forall \overline{K'} \neq \overline{K}$, то $\Gamma(S_i, K') < 0$, $\forall K' \neq K$. Поскольку $\Gamma(S_i, K) \geqslant 0$, отступ $\Delta(S_i, K) > 0$, и объект S_i распознаётся алгоритмом A правильно. ■

1.2.3. Классические логические алгоритмы распознавания и ранее построенные логические корректоры в рамках схемы голосования по корректным предикатам

Понятия корректного эл.кл., представительного набора и теста могут быть переформулированы на языке предикатов. Следующие два очевидных утверждения пока-

зывают, что алгоритмы голосования по корректным эл.кл. и голосования по тестам вписываются в схемы логического корректора общего вида.

Утверждение 1.2. *Эл.кл. (H, σ) корректен (является представительным набором) для класса K тогда и только тогда, когда предикат $[H(S) = \sigma]$ является корректным (представительным) для K. При этом функция $F(t_1) = t_1$ является корректирующей для набора эл.кл. $U = \big((H, \sigma)\big)$ относительно класса K.* ■

Утверждение 1.3. *Набор признаков H является тестом тогда и только тогда, когда для любого класса K и любого прецедента $S_i \in K$ предикат $[H(S_i) = H(S)]$ является представительным для K. При этом функций $F(t_1) = t_1$ является корректирующей для набора эл.кл. $U = \big((H, H(S_i))\big)$ относительно класса K.* ■

Ранее построенные логические корректоры также являются частными случаями логического корректора общего вида.

1. В логическом корректоре с произвольной корректирующей функцией производится голосование по представительным предикатам вида $[U(S_i) = U(S)]$, U — корректный для класса K набор эл.кл. (по старому понятию), S_i — прецедент из K.

2. В логическом корректоре МОН производится голосование по представительным предикатам вида $[U(S_i) \preccurlyeq U(S)]$, U — монотонный корректный для класса K набор эл.кл., S_i — прецедент из K.

3. В логическом корректоре АМОН производится голосование по корректным, предикатам вида $[U(S_i) \preccurlyeq U(S)]$, не являющихся представительными, где U — антимонотонный корректный для класса K набор эл.кл., S_i — прецедент из $\overline{K}$.

Понятия монотонного и антимонотонного корректного набора эл.кл. тесно связаны с понятиями теста, представительного и антипредставительного набора. Формально эта связь описывается следующими утверждениями.

Утверждение 1.4. *Пусть H — тест. Тогда для любого класса K существует монотонный корректный для K набор эл.кл. U такой, что для любого прецедента $S_i \in K$ выполняется тождество $[H(S_i) = H(S)] \equiv [U(S_i) \preccurlyeq U(S)]$, $\forall S \in M$.*

Доказательство. Зафиксируем класс K. Для каждого признака $x_j \in H$ и прецедента $S_i \in K$ построим одноранговый эл.кл. $(\{x_j\}, x_j(S_i))$. Составим из раз-

личных построенных эл.кл. набор U. Из конструкции набора эл.кл. U очевидно, что равенство $H(S_i) = H(S)$ выполняется тогда и только тогда, когда верно $U(S_i) \preccurlyeq U(S)$. Для любых прецедентов $S_i \in K$ и $S_t \notin K$ выполняется $[U(S_i) \preccurlyeq U(S_t)] = [H(S_i) = H(S_t)] = 0$, так как H — тест. Следовательно U — монотонный корректный для K набор эл.кл. ■

Аналогично доказываются следующие два утверждения.

Утверждение 1.5. *Пусть U — монотонный корректный для класса K набор эл.кл. Тогда для любого прецедента $S_i \in K$ существует представительный для K набор (H, σ) такой, что $[H(S) = \sigma] \equiv [U(S_i) \preccurlyeq U(S)]$, $\forall S \in M$.* ■

Утверждение 1.6. *Пусть U — антимонотонный корректный для класса K набор эл.кл. Тогда для любого прецедента $S_i \notin K$ существует антипредставительный для K набор (H, σ) такой, что $[H(S) = \sigma] \equiv [U(S_i) \preccurlyeq U(S)]$, $\forall S \in M$.* ■

Алгоритмы распознавания A_1 и A_2 будем называть *эквивалентными*, если они одинаково классифицируют все объекты из M, то есть $A_1(S) \equiv A_2(S)$. Из утверждений 1.4, 1.5 и 1.6 можно сделать 3 вывода.

1. Для любого тестового алгоритма существует эквивалентный ему логический корректор МОН.
2. Для любого логического корректора МОН существует эквивалентный ему алгоритм голосования по представительным наборам.
3. Для любого логического корректора АМОН существует эквивалентный ему алгоритм голосования по антипредставительным наборам.

1.3. Логический корректор POLAR с поляризуемой корректирующей функцией

Предлагается новый практический логический корректор POLAR, имеющий в качестве корректирующей поляризуемую булеву функцию. В данном корректоре в роли голосующих предикатов выступают так называемые поляризуемые предикаты. Построение этих предикатов сводится к поиску покрытий специальной булевой матрицы. Рассматривается задача поиска голосующих предикатов с наибольшей информативностью.

1.3.1. Поляризуемые предикаты

Пусть $U{=}((H_1,\sigma_1),\ldots,(H_d,\sigma_d))$ — набор эл.кл., G — набор прецедентов, $R{=}(r_1,\ldots,r_d)$ — набор бинарных отношений на $\{0,1\}$, например, r_1 — отношение «меньше или равно», $r_1(x,y)=[x{\leqslant}y]$; r_2 — отношение «больше или равно», $r_2(x,y)=[x{\geqslant}y]$; r_3 — отношение «равно», $r_3(x,y)=[x{=}y]$. Для бинарных векторов $\alpha=(\alpha_1,\ldots,\alpha_d)$ и $\beta=(\beta_1,\ldots,\beta_d)$ через $R(\alpha,\beta)$ обозначим конъюнкцию $r_1(\alpha_1,\beta_1)\wedge\ldots\wedge r_d(\alpha_d,\beta_d)$. Рассмотрим предикат

$$B_{(U,R,G)}(S)=\bigvee_{S_i\in G}R(U(S_i),U(S)). \tag{1.10}$$

Выясним условия, при которых предикат $B_{(U,R,G)}$ корректен для класса K.

Пусть G_1 и G_2 — множества объектов из M. Будем говорить, что набор эл.кл. U *отделяет* объекты из G_1 от объектов из G_2 с помощью набора бинарных отношений R, если не существует двух объектов $S'\in G_1$ и $S''\in G_2$, для которых выполняется равенство $R(U(S'),U(S''))=1$.

В частности, если набор эл.кл. U отделяет прецеденты из класса K от прецедентов из $\overline{K}$ с помощью набора бинарных отношений R, состоящего из бинарных отношений «равно» («меньше или равно»), то набор эл.кл. U является (монотонным) корректным для класса K.

Утверждение 1.7. *Пусть G — набор прецедентов класса K и набор эл.кл. $U{=}((H_1,\sigma_1),\ldots,(H_d,\sigma_d))$ отделяет объекты из G от прецедентов из $\overline{K}$ с помощью набора бинарных отношений $R{=}(r_1,\ldots,r_d)$. Тогда предикат $B_{(U,R,G)}(S)$ является корректным для K, и набор эл.кл. U является корректным для K с корректирующей функцией*

$$F_{(U,R,G)}(t_1,\ldots,t_d)=\bigvee_{S_i\in G}R(U(S_i),(t_1,\ldots,t_d)). \tag{1.11}$$

Доказательство. Из условия утверждения следует, что для любого прецедента $S_t\notin K$ выполняется $B_{(U,R,G)}(S_t)=0$. Следовательно, $B_{(U,R,G)}$ корректен для K. Так как предикат $F_{(U,R,G)}(U)=B_{(U,R,G)}$ корректен для K, набор эл.кл. U корректен для K и имеет корректирующую функцию $F_{(U,R,G)}$ по определению. ■

Пусть $U=((H_1,\sigma_1),\ldots,(H_d,\sigma_d))$ — набор эл.кл., $R=(r_1,\ldots,r_d)$ — набор бинарных отношений на $\{0,1\}$, G — набор прецедентов класса K. Корректный для K предикат $B_{(U,R,G)}$ будем называть *тупиковым*, если выполняются два условия:

1) для любого прецедента $S_i \in K \setminus G$ предикат $B_{(U,R,G\cup\{S_i\})}$ не является корректным для K;

2) для набор эл.кл. $U' = ((H_1,\sigma_1),\ldots,(H_{j-1},\sigma_{j-1}),(H_{j+1},\sigma_{j+1}),\ldots(H_d,\sigma_d))$ и набора отношений $R' = (r_1,\ldots,r_{j-1},r_{j+1},\ldots,r_d)$, $j \in \{1,\ldots,d\}$, предикат $B_{(U',R',G)}(S)$ не является корректным для K.

Обозначим через $\mathcal{B}_K$ множество корректных для K предикатов вида (1.10). Подмножество $\mathcal{B}_K$, состоящее из тупиковых предикатов, обозначим через $\mathcal{B}_K^*$.

Заметим, что если информативность предикатов из $\mathcal{B}_K$ оценивать функционалом

$$P(B,K) = \sum_{S_i \in K} B(S_i),$$

то максимум $P(B,K)$ на $\mathcal{B}_K$ совпадает с его максимумом на $\mathcal{B}_K^*$. В этом смысле целесообразно рассматривать только тупиковые предикаты.

Будем говорить, что предикаты $B_{(U,R,G)}$ и $B_{(U',R',G')}$ *эквивалентны*, если они задают одинаковые отображения из M в $\{0,1\}$, $B_{(U,R,G)}(S) = B_{(U',R',G')}(S)$, $\forall S{\in}M$. Множество всех предикатов вида (1.10) разбивается на классы эквивалентности.

Утверждение 1.8. *Для любого предиката $B_{(U,R,G)}$ существует эквивалентный предикат $B_{(U',R',G)}$ такой, что набор U' состоит из эл.кл. набора U и каждое бинарное отношение набора R' принадлежит множеству $\mathcal{R}^*{=}\{[x{\leqslant}y],[x{\geqslant}y],[x{\vee}y],[{\neg}x{\vee}{\neg}y]\}$.*

Доказательство. Заметим, что каждое отношение из $\mathcal{R}^*$ принимает нулевое значение всего лишь на одной паре значений аргументов. Например, $[x \leqslant y] = 0$, тогда и только тогда, когда $x = 1$ и $y = 0$.

Построим требуемые наборы R' и U'. Каждое отношение r_j в R заменим на набор отношений $r'_1,\ldots,r'_u$ из $\mathcal{R}^*$ таких, что $r_j(x,y) = r'_1(x,y){\wedge}\ldots{\wedge}r'_u(x,y)$. Например, если $r_j(x,y) = [x = y]$, то можно взять $r'_1(x,y) = [x \leqslant y]$ и $r'_2(x,y) = [x \geqslant y]$. Каждому отношению r'_v, $v \in \{1,\ldots,u\}$, сопоставим эл.кл. (H'_v,σ'_v), совпадающий с эл.кл. (H_j,σ_j). Полученные в результате такого построения наборы R' и U' и будут определять предикат $B_{(U',R',G)}$, эквивалентный предикату $B_{(U,R,G)}$. ■

Фактически из утверждения 1.8 следует, что при построении предикатов вида (1.10) можно использовать только отношения из $\mathcal{R}^*$.

Корректный для K предикат $B_{(U,R,G)}$ будем называть *монотонным (поляризуемым)*, если функция $F_{(U,R,G)}$ является монотонной (поляризуемой).

Утверждение 1.9. *Пусть $B_{(U,R,G)}$ — корректный для K предикат. Тогда выполняются следующие условия.*

1. *Если каждое отношение набора R принадлежит $\mathcal{R}^*$, то предикат $B_{(U,R,G)}$ является поляризуемым.*

2. *Если каждое отношение набора R совпадает либо с $[x \leqslant y]$, либо с $[x \vee y]$, то предикат $B_{(U,R,G)}$ является монотонным.*

Доказательство. Заметим, что $[x \leqslant y] = [\neg x \vee y]$ и $[x \geqslant y] = [x \vee \neg y]$.

1. Использование отношений из $\mathcal{R}^*$ гарантирует, что каждая переменная булевой функции $F_{(U,R,G)}$ входят в дизъюнктивную нормальную форму (1.11) либо только с отрицанием, либо только без отрицания. Следовательно $F_{(U,R,G)}$ является поляризуемой.

2. С отношениями $[x \leqslant y]$ и $[x \vee y]$ в дизъюнктивную нормальную форму (1.11) ни одна переменная не будет входить с отрицанием, что эквивалентно монотонности корректирующей функции $F_{(U,R,G)}$. ■

Утверждение 1.10. *Пусть G — непустой набор прецедентов класса K и предикат $B_{(U,R,G)}$ корректен для K. Если каждое отношение набора R принадлежит множеству $\{[x \leqslant y], [x \geqslant y], [x = y]\}$, то предикат $B_{(U,R,G)}$ является представительным для K.*

Доказательство. Поскольку R состоит из отношений $\{[x \leqslant y], [x \geqslant y], [x = y]\}$, выполняется $R(U(S), U(S)) = 1$ для любого $S \in M$. Тогда $B_{(U,R,G)}(S_i) = 1$ для любого $S_i \in G$. Так как $G \neq \varnothing$, предикат $B_{(U,R,G)}$ представителен для K. ■

Из утверждений 1.9 и 1.10 следует, что корректный для K предикат $B_{(U,R,G)}$ такой, что $G \subset K$, $G \neq \varnothing$ и каждое отношение набора R совпадает либо с $[x \leqslant y]$, либо с $[x \geqslant y]$, является поляризуемым и представительным для K. Обозначим множество таких предикатов через $\mathcal{P}_K$ и множество $\mathcal{B}_K^* \cap \mathcal{P}_K$ через $\mathcal{P}_K^*$.

Проиллюстрируем на примере с модельной задачей преимущество использования поляризуемых предикатов.

Пример 1.3. Рассмотрим задачу распознавания с двумя классами и двумя признаками, изображенную на рисунке 1.1. Рассмотрим два набора эл.кл.

1. Набор $U_1 = ([x_2 = 0],\ [x_1 = 0],\ [x_1 = 1],\ [x_1 = 2],\ [x_1 = 3],\ [x_1{=}4])$ является монотонным корректным для класса K_1, то есть отделяет прецеденты из K_1

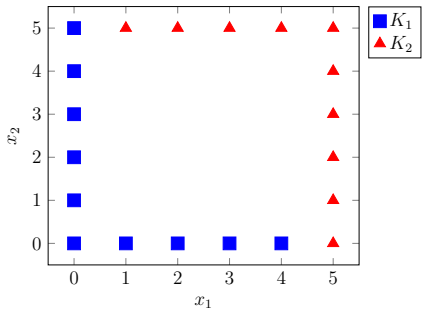


Рисунок 1.1: Модельная задача распознавания из примера 1.3

от прецедентов из K_2 с помощью набора отношений $R_1{=}([x{\leqslant}y],\ldots,[x{\leqslant}y])$. Набор U_1 имеет наименьшую мощность среди монотонных корректных для K_1 наборов эл.кл.

2. Набор $U_2 = ([x_1 = 5],\ [x_2 = 0],\ [x_1 = 0])$ отделяет прецеденты из K_1 от прецедентов из K_2 с помощью набора отношений $R_2{=}([x{\geqslant}y],[x{\leqslant}y],[x{\leqslant}y])$. Набор U_2 имеет наименьшую мощность среди корректных наборов эл.кл. с поляризуемой корректирующей функцией.

Выпишем предикаты $B_{(U_j,R_j,\{S_i\})}, S_i \in K_1,\ j = 1,2$, и вычислим информативность этих предикатов.

1. Монотонные предикаты: $[x_1 = 0]$, $[x_1 = 0 \wedge x_2 = 0]$, $[x_1 = 1 \wedge x_2 = 0]$, $[x_1 = 2 \wedge x_2 = 0]$, $[x_1 = 3 \wedge x_2 = 0]$,$[x_1 = 4 \wedge x_2 = 0]$. Первый предикат имеет информативность 6, информативности остальных равны 1.

2. Поляризуемые предикаты: $[x_1 \neq 5 \wedge x_2 = 0]$ и $[x_1 = 0]$. Первый предикат имеет информативность 5, второй — 6.

Видно, что поляризуемые предикаты более лаконичны и имеют высокую информативность. Это становится возможным, благодаря тому, что с помощью одновременного использования отношений $[x \leqslant y]$ и $[x \geqslant y]$ удается одним предикатом проверить как наличие, так и отсутствие некоторого признакового подописания у распознаваемого объекта. ■

1.3.2. Алгоритм голосования по поляризуемым предикатам

Опишем логический корректор POLAR, основанный на голосовании по поляризуемым предикатам.

На этапе обучения для каждого класса K формируются два семейства Z_K и $Z_{\overline{K}}$, $Z_K \subseteq \mathcal{P}_K$, $Z_{\overline{K}} \subseteq \mathcal{P}_{\overline{K}}$. Предикату B приписывается вес $\alpha_B > 0$.

Распознавание осуществляется взвешенным голосованием по предикатам, построенным на этапе обучения. Возможны два режима распознавания: базовый и аддитивный.

1. *В базовом режиме* при распознавании объекта S для каждого класса K вычисляется оценка $\Gamma(S, K)$ принадлежности объекта S классу K, имеющая вид

$$\Gamma(S, K) = \sum_{B \in Z_K} \alpha_B B(S) - \sum_{B \in Z_{\overline{K}}} \alpha_B B(S),$$

   то есть распознавание осуществляется аналогично логическому корректору общего вида.

2. *В аддитивном режиме* для распознаваемого объекта S и каждого построенного предиката $B_{(U,R,G)}$ вычисляется оценка

$$\gamma(S, B_{(U,R,G)}) = \frac{1}{|G|} \sum_{S_i \in G} R(U(S_i), U(S)).$$

   Затем для каждого класса K вычисляется оценка $\Gamma(S, K)$ принадлежности объекта S классу K, имеющая вид

$$\Gamma(S, K) = \sum_{B \in Z_K} \alpha_B \gamma(S, B) - \sum_{B \in Z_{\overline{K}}} \alpha_B \gamma(S, B).$$

Основная задача этапа обучения логического корректора POLAR — поиск информативных корректных предикатов из $\mathcal{P}_K$.

1.3.3. Сведение задачи построения поляризуемых предикатов к поиску покрытий булевой матрицы

В работе [15] построение корректных наборов эл.кл. сводится к поиску покрытий булевой матрицы, построенной специальным образом по обучающей выборке. В данном подразделе выполняется аналогичное сведение построения предикатов из $\mathcal{P}_K$ к поиску покрытия булевой матрицы.

Пусть $L = \|a_{ij}\|$ — булева матрица размера $m \times n$. Говорят, что столбец с номером j покрывает строку с номером i булевой матрицы L, если $a_{ij} = 1$. *Покрытием*

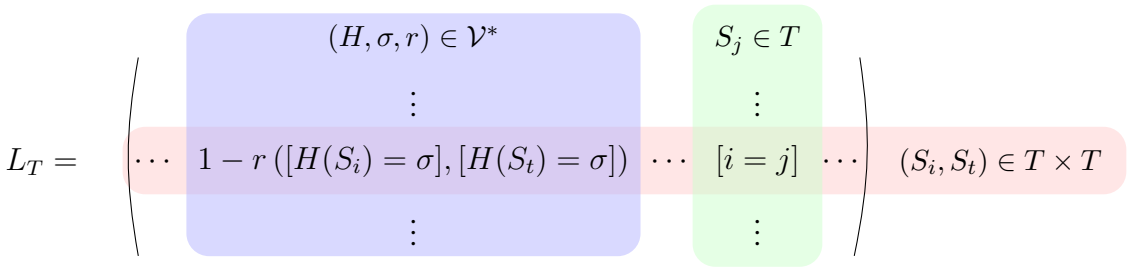


Рисунок 1.2: Схема построения матрицы сравнения L_T.

булевой матрицы L называется набор столбцов J такой, что каждую строку матрицы L покрывает хотя бы один столбец из J. Покрытие J матрицы L называется *неприводимым*, если любое его собственное подмножество не является покрытием матрицы L. Задача перечисления всех неприводимых покрытий булевой матрицы называется *дуализацией*. Подходы к её решению подробно изложены в главе 3.

Обозначим через $\mathcal{V}^*$ множество троек (H, σ, r), где (H, σ) — эл.кл. и r — одно из отношений $[x \leqslant y]$ или $[x \geqslant y]$.

Построим булеву матрицу L_T по следующему правилу. Каждой строке матрицы L_T сопоставим пару обучающих объектов $(S_i, S_t) \in T \times T$. Столбцы матрицы L_T будут иметь один из двух типов. Каждому столбцу первого типа сопоставим тройку $(H, \sigma, r) \in \mathcal{V}^*$. Каждому столбцу второго типа — прецедент $S_j \in T$. Элемент матрицы L_T, расположенный на пересечении строки (S_i, S_t) и столбца (H, σ, r), равен $1 - r\left([H(S_i) = \sigma], [H(S_t) = \sigma]\right)$. Элемент матрицы L_T, расположенный на пересечении строки (S_i, S_t) и столбца S_j, равен $[i = j]$. Матрицу, построенную по указанному правилу, принято называть *матрицей сравнения*. Наглядно способ её построения изображен на рис. 1.2.

Пусть K — класс или дополнение класса, $K \in \{K_1, \ldots, K_l, \overline{K}_1, \ldots, \overline{K}_l\}$. Через L_K обозначим подматрицу $L_T\big((T \cap K) \times (T \setminus K), \mathcal{V}^* \cup (T \cap K)\big)$ (здесь и далее через $L(R, C)$ обозначается подматрица матрицы L, составленная из её строк R и столбцов C).

Утверждение 1.11. *Пусть $U = ((H_1, \sigma_1), \ldots, (H_d, \sigma_d))$ — набор эл.кл., $R = (r_1, \ldots, r_d)$ — набор отношений из $\{[x \leqslant y], [x \geqslant y]\}$ и G — набор прецедентов класса K.*

Предикат $B_{(U,R,G)}$ является (тупиковым) корректным для K тогда и только тогда, когда набор столбцов $J = ((T \cap K) \setminus G) \cup \{(H_1, \sigma_1, r_1), \ldots, (H_d, \sigma_d, r_d)\}$ является (неприводимым) покрытием матрицы L_K.

Доказательство. Корректность предиката $B_{(U,R,G)}$ для K по определению означает, что выполняется $B_{(U,R,G)}(S_t) = 0, \forall S_t \in T \setminus K$. Поскольку верны тождества

$$B_{(U,R,G)}(S) = \bigvee_{S_i \in G} R(U(S_i), U(S)) = \bigvee_{S_j \in K} [S_j \notin (T \cap K) \setminus G]\ R(U(S_j), U(S)),$$

корректность предиката $B_{(U,R,G)}$ эквивалентна условию

$$\bigvee_{S_j \in K} \bigvee_{S_t \notin K} [S_j \notin (T \cap K) \setminus G]\ R(U(S_j), U(S_t)) = 0. \tag{1.12}$$

Отрицая левую и правую часть равенства (1.12), получаем условие

$$\begin{aligned} \bigwedge_{S_j \in K} \bigwedge_{S_t \notin K} \quad & \big([S_j \in (T \cap K) \setminus G] \vee \\ \vee\ & \neg r_1([H_1(S_j) = \sigma_1], [H_1(S_t) = \sigma_1]) \vee \\ \vee\ & \ldots \vee \neg r_d([H_d(S_j) = \sigma_d], [H_d(S_t) = \sigma_d])\big) = 1, \end{aligned}$$

которое равносильно тому, что набор столбцов J покрывает матрицу L_K.

Из определения тупикового корректного предиката легко выводится, что $B_{(U,R,G)}$ принадлежит $\mathcal{P}_K^*$ тогда и только тогда, когда при удалении любого столбца из J получается набор столбцов, не являющийся покрытием L_K, то есть J — неприводимое покрытие L_K. ■

1.3.4. Поиск поляризуемых предикатов с наибольшей информативностью

В реальных задачах число покрытий матрицы L_K очень велико. При этом далеко не каждое покрытие L_K соответствует «хорошему» предикату. Далее вводятся функционалы информативности голосующих предикатов, и поиск предикатов с наибольшей информативностью сводится с специальным дискретным оптимизационным задачам.

Пусть B — предикат на множестве объектов M и K — класс. С каждым объектом S_i обучающей выборки T свяжем неотрицательный вес w_i, характеризующий цену ошибки на объекте S_i. Обозначим $\vec{w} = (w_1, \ldots, w_m)$. Введём зависящие от

взвешенной выборки $(T, \vec{w})$ функционалы

$$P(B,K) = \sum_{S_i \in K} w_i B(S_i), \quad N(B,K) = \sum_{S_i \in \overline{K}} w_i B(S_i).$$

Разность $P(B,K) - N(B,K)$ будем называть *информативностью* предиката B для K и обозначать через $I(B,K)$. Заметим, что функционалы, аналогичные $I(B,K)$, часто используются для оценки распознающей способности логических закономерностей в распознавании (см. [67]).

В базовом режиме работы логического корректора POLAR информативность предиката $B_{(U,R,G)}$ будем оценивать функционалом $I(B_{(U,R,G)}, K)$. В аддитивном режиме более адекватную оценку информативности предиката $B_{(U,R,G)}$ даёт функционал

$$\begin{aligned} \hat{I}(B_{(U,R,G)}, K) &= \hat{P}(B_{(U,R,G)}, K) - \hat{N}(B_{(U,R,G)}, K), \quad \text{где} \\ \hat{P}(B_{(U,R,G)}, K) &= \sum_{S \in G} P(B_{(U,O,\{S\})}, K), \quad \hat{N}(B_{(U,R,G)}, K) = \sum_{S \in G} N(B_{(U,O,\{S\})}, K). \end{aligned}$$

Пусть G^+ — набор прецедентов класса K и G^- — набор прецедентов из $\overline{K}$. Обозначим через $\mathcal{P}_K(G^+, G^-)$ семейство предикатов $B_{(U,R,G)}$ таких, что $G \subseteq G^+$ и набор эл.кл. U отделяет прецеденты из G от прецедентов из G^- с помощью набора отношений R.

Аналогично утверждению 1.11 доказывается

Утверждение 1.12. *Пусть G^+ — набор прецедентов класса K, G^- — набор прецедентов из $\overline{K}$, $G \subseteq G^+$, $\overline{G} = G^+ \setminus G$, $U = ((H_1, \sigma_1), \ldots, (H_d, \sigma_d))$ — набор эл.кл., $R = (r_1, \ldots, r_d)$ — набор отношений, $V = \{(H_1, \sigma_1, r_1), \ldots, (H_d, \sigma_d, r_d)\}$. Предикат $B_{(U,R,G)}$ принадлежит семейству $\mathcal{P}_K(G^+, G^-)$ тогда и только тогда, когда набор столбцов $V \cup \overline{G}$ матрицы L_K является покрытием подматрицы $L_K(G^+ \times G^-, \mathcal{V}^* \cup G^+)$.* ■

Рассмотрим задачи построения предиката $B_{(U,R,G)}$ из $\mathcal{P}_K(G^+, G^-)$, обладающего максимальной информативностью относительно взвешенной выборки $(T, \vec{w})$.

Задача 1.1.

$$I(B,K) \underset{B \in \mathcal{P}_K(G^+,G^-)}{\longrightarrow} \max.$$

Задача 1.2.

$$\hat{I}(B,K) \underset{B \in \mathcal{P}_K(G^+,G^-)}{\longrightarrow} \max.$$

Сформулируем две дискретные оптимизационные задачи, являющиеся специальными разновидностями задачи о поиске покрытий булевой матрицы. Обозначим через $r_L(J)$ набор строк матрицы L, покрытых набором J, и через $\mathcal{C}(L)$ набор покрытий булевой матрицы L.

Задача 1.3 (Поиск набора столбцов, покрывающего оптимальную комбинацию матриц). Пусть даны булевы матрицы $L_0, L_1, \ldots, L_d$ и ненулевые веса $\alpha_1, \ldots, \alpha_d$. Каждая матрица имеет n столбцов. Требуется найти (неприводимое) покрытие J матрицы L_0 такой, что сумма весов матриц, не покрытых J, максимальна, то есть

$$\sum_{J \notin \mathcal{C}(L_i)} \alpha_i \underset{J \in \mathcal{C}(L_0)}{\longrightarrow} \max.$$

Задача 1.4 (Поиск набора столбцов, покрывающего оптимальную комбинацию строк). Пусть даны две булевы матрица L_0 и L' с n столбцами. Для каждой строки i матрицы L' задан ненулевой вес β_i. Требуется найти (неприводимое) покрытие J матрицы L_0 такое, что сумма весов строк матрицы L', не покрытых J, максимальна, то есть

$$\sum_{i \notin r_{L'}(J)} \beta_i \underset{J \in \mathcal{C}(L_0)}{\longrightarrow} \min.$$

Покажем, что задача 1.1 сводится к задаче 1.3, а задача 1.2 — к задаче 1.4.

Пусть $J_0 = \mathcal{V}^* \cup G^+$, $L_0 = L_T(G^+ \times G^-, J_0)$, $L_i = L_T(G^+ \times \{S_i\}, J_0)$, $S_i \in T \setminus G^-$, и $L' = L_T(G^+ \times (T \setminus G^-), J_0)$. Справедливо

Утверждение 1.13. *Пусть $U = ((H_1, \sigma_1), \ldots, (H_d, \sigma_d))$ — набор эл.кл., $R = (r_1, \ldots, r_d)$ — набор отношений, G^+ — набор прецедентов класса K, G^- — набор прецедентов из $\overline{K}$, предикат $B_{(U,R,G)} \in \mathcal{P}_K(G^+, G^-)$ и $J = (G^+ \setminus G) \cup \{(H_1, \sigma_1, r_1), \ldots, (H_d, \sigma_d, r_d)\}$. Тогда*

$$P(B_{(U,R,G)}, K) = \sum_{S_i \in K} w_i[J \notin \mathcal{C}(L_i)], \quad N(B_{(U,R,G)}, K) = \sum_{S_i \notin K} w_i[J \notin \mathcal{C}(L_i)].$$

Доказательство. Первого равенства следует из простой цепочки тождеств:

$$\begin{aligned} P(B_{(U,R,G)},K) &= \sum_{S_i\in K} w_i B_{(U,R,G)}(S_i) = \sum_{S_i\in K} w_i \bigvee_{S_j\in G} R(U(S_j),U(S_i)) = \\ &= \sum_{S_i\in K} w_i \bigvee_{S_j\in G^+} [S_j \notin G^+ \setminus G]\; R(U(S_j),U(S_i)) = \\ &= \sum_{S_i\in K} w_i \Big(1 - \bigwedge_{S_j\in G^+} [S_j \in G^+ \setminus G] \vee\; \neg R(U(S_j),U(S_i))\Big) = \\ &= \sum_{S_i\in K} w_i [J \notin \mathcal{C}(L_i)]. \end{aligned}$$

Равенство для $N(B_{(U,R,G)},K)$ доказывается аналогично. ■

Аналогично утверждению 1.13 доказывается

Утверждение 1.14. *Пусть* $U = ((H_1,\sigma_1),\ldots,(H_d,\sigma_d))$ *— набор эл.кл.,* $R = (r_1,\ldots,r_d)$ *— набор отношений,* G^+ *— набор прецедентов класса* K*,* G^- *— набор прецедентов из* $\overline{K}$*, предикат* $B_{(U,R,G)} \in \mathcal{P}_K(G^+,G^-)$ *и* $J = (G^+ \setminus G) \cup \{(H_1,\sigma_1,r_1),\ldots,(H_d,\sigma_d,r_d)\}$*. Тогда*

$$\begin{aligned} \hat{P}(B_{(U,R,G)},K) &= \sum_{(S_j,S_i)\notin r_{L'}(J)} w_i [S_i \in K], \\ \hat{N}(B_{(U,R,G)},K) &= \sum_{(S_j,S_i)\notin r_{L'}(J)} w_i [S_i \notin K]. \end{aligned}$$

■

Каждой матрице L_i, $S_i \in T\setminus G^-$, и каждой строке (S_j,S_i) матрицы L' припишем вес, равный

$$\begin{cases} w_i, & S_i \in K, \\ -w_i, & S_i \notin K. \end{cases}$$

Из утверждений 1.12 и 1.13 следует, что покрытие L_0, покрывающее оптимальную комбинацию взвешенных матриц L_i, $S_i \in T \setminus G^-$, даёт решение задачи 1.1. Аналогично, из утверждений 1.12 и 1.14 следует, что покрытие L_0, покрывающее оптимальную комбинацию взвешенных строк матрицы L' даёт решение задачи 1.2.

Аналоги задачи 1.3 автору не известны. Задача 1.4 обобщает ряд известных задач, однако её исследования в приведённой постановке не проводились.

Задача 1.5 (Red-Blue Set Cover Problem (RBSC)). Простой вариант задачи RBSC формулируется следующим образом [45,68]. Входом являются множество «красных»

элементов R, множество «синих» элементов B и набор $\mathcal{D}$ подмножеств множества $R \cup B$. Говорят, что элемент $e \in R \cup B$ покрыт набором $\mathcal{D}' \subseteq \mathcal{D}$, если e принадлежит хотя бы одному множеству из $\mathcal{D}'$. Обозначим через $\mathcal{C}(\mathcal{D}')$ множество элементов, покрытых набором $\mathcal{D}'$. Требуется найти подмножество $\mathcal{D}'$ множества $\mathcal{D}$, которое покрывает все синие элементы и как можно меньше красных элементов, то есть

$$|R \cap \mathcal{C}(\mathcal{D})| \underset{\mathcal{D}' \subseteq \mathcal{D}: B \subseteq \mathcal{C}(\mathcal{D}')}{\rightarrow} \min .$$

В [45] также рассматривается «взвешенный» вариант RBSC. Во взвешенном варианте RBSC каждому красному элементу присваивается положительный вес и требуется минимизировать сумму весов покрытых красных элементов.

В случае, когда строки матрицы L имеют отрицательный веса, задача 1.4 эквивалентна RBSC, в которой синими элементами являются строки матрицы L_0, красными — строки L', и вес каждого красного элемента равен весу соответствующей строки L', взятому с противоположным знаком.

Задача 1.6 (Positive–Negative Partial Set Cover Problem (±PSC)). Входом, аналогично RBSC, являются множество «красных» (отрицательных) элементов R, множество «синих» (положительных) элементов B и набор $\mathcal{D}$ подмножеств множества $R \cup B$. Требуется найти подмножество $\mathcal{D}'$ множества $\mathcal{D}$, которое покрывает как можно больше синих элементов и как можно меньше красных, то есть

$$|R \cap \mathcal{C}(\mathcal{D})| - |B \cap \mathcal{C}(\mathcal{D})| \underset{\mathcal{D}' \subseteq \mathcal{D}: B \subseteq \mathcal{C}(\mathcal{D}')}{\rightarrow} \min .$$

В случае, когда каждая строка матрицы L' имеет вес ± 1 и число строк L_0 равно 0, задача 1.4 эквивалентна ±PSC, в которой красными элементами являются строки матрицы L', имеющие вес -1, синими — строки L' с весом 1. Задача ±PSC изучается в [46].

В настоящей работе для решения задач 1.3 и 1.4 использовался метод ветвей и границ на базе алгоритмов дуализации из главы 3. Сложность такого варианта решения не исследовалась. Однако очевидно, что она существенно зависит от размеров входных матриц. В следующей главе предлагается методика, позволяющая при обучении логических корректоров использовать только часть строк и столбцов матрицы сравнения.

ГЛАВА 2

Методы повышения эффективности логических корректоров

Во данной главе разрабатывается методика повышения скорости обучения и качества распознавания логических корректоров. Семейства голосующих предикатов строятся итеративно по принципу *бустинга*. Поиск голосующих предикатов осуществляется в рамках *локальных базисов классов* — предварительно формируемых корректных наборов, состоящих из информативных эл.кл. Эффективность предложенной методики тестируется на реальных данных.

2.1. Итеративное формирование семейств голосующих предикатов по принципу бустинга

Предлагается семейства голосующих предикатов логического корректора POLAR формировать итеративно по принципу бустинга [41, 42]. Бустинг является универсальным методом построения корректных алгоритмов распознавания в виде линейных комбинаций базовых, необязательно корректных, распознающих алгоритмов. Метод бустинга в последние два десятилетия довольно быстро и плодотворно развивается. Модели бустинг-алгоритмов различаются функциями потерь, методами настройки весов и оптимизации отступов обучающих объектов. Особенности различных моделей достаточно полно освещены в обзоре [69].

В данной работе рассматривается исторически первая, в некотором смысле простейшая модель бустинг-алгоритмов — AdaBoost, позволяющая решить две проблемы ранее построенных логических корректоров. Во-первых, повышается качество распознавания за счёт настройки весов голосующих предикатов и увеличения

диверсификации семейств предикатов. Во-вторых, снижаются временные затраты обучения за счёт того, что поиск предикатов осуществляется не по всей матрице сравнения, а лишь по подматрице, составленной из части её строк.

Заметим, что первые успешные попытки применения бустинга при построении логических корректоров были предприняты в работах [16, 17]. Фактически, в этих работах использовался симбиоз бустинга и предложенного в [70] генетического алгоритма поиска корректного набора эл.кл. с распознающей способностью, близкой к максимальной. Эксперименты на прикладных задачах показали, что логические корректоры, использующие и бустинг, и генетический алгоритм, работают лучше ранее построенных логических корректоров, основанных только на генетическом алгоритме.

2.1.1. Понятия и обозначения, необходимые для описания бустинг-алгоритма

Рассмотрим логический корректор POLAR, который работает в базовом режиме. Для аддитивного режима применимы все приводимые ниже рассуждения с незначительными изменениями.

Пусть выполнено t, $t \geqslant 0$, итераций, $S_i \in T$ и K — класс. Введём обозначения: A_t — логический корректор, голосующий по предикатам, построенным за t итераций (голосующие семейства логического корректора A_0 пусты), $\Gamma_t(S_i, K)$ — оценка за отнесение объекта S_i к классу K, вычисляемая по семействам голосующих предикатов логического корректора A_t, y_i — номер класса, которому принадлежит S_i, и $M_t(S_i, K) = \Gamma_t(S_i, K_{y_i}) - \Gamma_t(S_i, K)$.

Для числа ошибок и отказов алгоритма A_t на обучении справедливо неравенство

$$Q(A_t) = \sum_{i=1}^{m} [A_t(S_i) \neq y_i] \leqslant \sum_{y=1}^{l} \sum_{S_i \notin K_y} [M_t(S_i, K_y) \leqslant 0].$$

Заметим, что в случае $l = 2$ неравенство обращается в равенство, и в случае $t = 0$ правая часть неравенства равна $m(l-1)$.

Построим логический корректор A_{t+1}, не меняя предикаты и их веса, найденные на итерациях $1, \ldots, t$. На итерации $t+1$ по некоторому правилу выберем класс K, построим поляризуемый предикат B и настроим его вес α_B так, чтобы при добавлении B в Z_K наилучшим образом компенсировались ошибки ранее построен-

ных предикатов логического корректора A_t. При этом желательно минимизировать суммарные потери $Q(A_{t+1})$.

В модели AdaBoost предлагается использовать гладкую монотонную функцию *потерь* $\mathcal{L}(x) = e^{-x}$, ограничивающую сверху функцию $f(x) = [x \leqslant 0]$, и решать оптимизационную задачу

$$\hat{Q}(A_{t+1}) = \sum_{y=1}^{l} \sum_{S_i \notin K_y} \mathcal{L}(M_{t+1}(S_i, K_y)) \to \min.$$

Поскольку, очевидно, выполняется неравенство $Q(A_{t+1}) \leqslant \hat{Q}(A_{t+1})$, справедлива оценка $\min Q(A_{t+1}) \leqslant \min \hat{Q}(A_{t+1})$.

Заметим, что для пары (S_i, K_y), $y \in \{1, \ldots, l\}$, $S_i \notin K_y$, такой, что $K_y \neq K$ и $K_{y_i} \neq K$, справедливо равенство $M_t(S_i, K_y) = M_{t+1}(S_i, K_y)$. Нетрудно видеть, что для остальных пар выполняется

$$M_{t+1}(S_i, K_y) = \begin{cases} M_t(S_i, K_y) + \alpha_B B(S_i), & S_i \in K, \\ M_t(S_i, K_y) - \alpha_B B(S_i), & S_i \notin K. \end{cases}$$

Сопоставим каждой паре (S_i, K) вес

$$w_t(S_i, K) = \frac{1}{\hat{Q}(A_t)} \begin{cases} \sum_{K_y \neq K} \exp(-M_t(S_i, K_y)), & S_i \in K, \\ \exp(-M_t(S_i, K)), & S_i \notin K \end{cases}$$

Заметим, что, если прецедент S_i принадлежит классу K, то вес $w_t(S_i, K)$ характеризует «трудность отделения» объекта S_i от прецедентов из $\overline{K}$ логическим корректором A_t, иначе вес $w_t(S_i, K)$ указывает насколько «трудно» прецедент S_i отличить от прецедентов класса K.

Обозначим $P_t(B, K) = \sum_{S_i \in K} w_t(S_i, K) B(S_i)$, $N_t(B, K) = \sum_{S_i \notin K} w_t(S_i, K) B(S_i)$. В результате несложных преобразований получим

$$\hat{Q}(A_{t+1}) = \hat{Q}(A_t)\big(1 + (e^{-\alpha_B} - 1)P_t(B, K) + (e^{\alpha_B} - 1)N_t(B, K)\big). \qquad (2.1)$$

Зафиксируем B и будем считать, что выполняется условие $P_t(B, K) > N_t(B, K) > 0$. Легко проверить, что минимальное значение $\hat{Q}(A_{t+1})$ по α_B достигается в точке

$$\alpha_B = \frac{1}{2} \ln \frac{P_t(B, K)}{N_t(B, K)}. \qquad (2.2)$$

После подстановки (2.2) в (2.1) становится видно, что минимум $\hat{Q}(A_{t+1})$ достигается на предикате, максимизирующем значение функционала $J_t(B,K)=\sqrt{P_t(B,K)}-\sqrt{N_t(B,K)}$. Заметим, что $\sqrt{2}J_t(B,K) \geqslant I_t(B,K)$, где $I_t(B,K) = P_t(B,K) - N_t(B,K)$, то есть $\max J_t(B,K) \geqslant \frac{1}{\sqrt{2}} \max I_t(B,K)$.

Вес, задаваемый формулой (2.2), не определен для корректного предиката B, так как $N_t(B,K) = 0$. Немного подправим формулу (2.2). Обозначим

$$\begin{aligned} N_t^*(B,K) &= \begin{cases} N_t(B,K), & N_t(B,K) > 0, \\ \frac{1}{2m}, & \text{иначе}, \end{cases} \\ J_t^*(B,K) &= \sqrt{P_t(B,K)} - \sqrt{N_t^*(B,K)}. \end{aligned}$$

Если $J_t^*(B,K) > 0$, то определен и положителен вес, вычисляемый по формуле

$$\alpha_B = \frac{1}{2} \ln \frac{P_t(B,K)}{N_t^*(B,K)}. \tag{2.3}$$

Рассмотрим семейство поляризуемых предикатов $\mathcal{P}_K(G^+, G^-)$. В этом семействе всегда существует предикат, для которого значение функционала J_t^* не меньше, чем для предиката $B_{bad}(S) = [S \in G^+ \cup (T \setminus K \setminus G^-)]$. Обозначим через $W_t(G^+, G^-, K)$ значение функционала $J_t^*(B_{bad}, K)$.

2.1.2. Бустинг-алгоритм обучения логического корректора POLAR

Далее описывается бустинг-алгоритм формирования голосующих семейств предикатов логического корректора POLAR, работающего в базовом режиме.

Параметры: t_{max} — число итераций, $\delta > 0$ — параметр выбора пространства поиска предикатов.

При инициализации взять $Z_{K_1} = \ldots = Z_{K_l} = Z_{\overline{K}_1} = \ldots = Z_{\overline{K}_l} = \varnothing$.

Пусть произведено t, $t \geqslant 0$, итераций. На итерации $t+1$ выполняется следующее.

1. Для каждого класса K и каждого прецедента S_i вычислить вес $w_t(S_i, K)$.
2. Выбрать класс K и семейство поляризуемых предикатов $\mathcal{P}_K(G^+, G^-)$ такие, что $W_t(G^+, G^-, K) > \delta$.
3. Найти в $\mathcal{P}_K(G^+, G^-)$ предикат B с наибольшей информативностью $I_t(B, K)$.
4. По формуле (2.3) вычислить вес α_B.

Алгоритм 1 Выбор области поиска поляризуемых предикатов

1: **ПРОЦЕДУРА** `SpecifySearchSpace(`t, δ`)`

Параметры: t — число выполненных итераций; $\delta > 0$ — параметр выбора семейства поляризуемых предикатов;

Выход: либо семейство поляризуемых предикатов $\mathcal{P}_K(G^+, G^-)$, удовлетворяющее условию $W_t(G^+, G^-, K) > \delta$, либо семейство поляризуемых предикатов $\mathcal{P}_K$;

2: инициализировать $\mathbb{K} := \{K \in \{K_1, \ldots, K_l\} \colon W_t(T \cap K, T \setminus K, K) > \delta\}$;
3: **если** $\mathbb{K} = \varnothing$ **то**
4: **вернуть** $\mathcal{P}_K$, где класс K имеет наибольшее значение $W_t(T \cap K, T \setminus K, K)$;
5: выбрать случайный класс K из распределения вероятностей $p_K = \frac{1}{\Pi} W_t(T \cap K, T \setminus K, K)$, $K \in \mathbb{K}$, где $\Pi = \sum_{K \in \mathbb{K}} W_t(T \cap K, T \setminus K, K)$;
6: упорядочить прецеденты $T \cap K$ и $T \setminus K$ по убыванию весов $w_t(S_i, K)$;
7: найти числа r_1 и r_2 такие, что
1) набор G^+, состоящий из первых r_1 объектов упорядоченного $T \cap K$ и набор G^-, состоящий из первых r_2 объектов упорядоченного $T \setminus K$, удовлетворяют условию $W_t(G^+, G^-, K) > \delta$ и
2) произведение $r_1 r_2$ минимально;
8: в качестве G^+ взять первые r_1 объектов упорядоченного $T \cap K$;
9: в качестве G^- взять первые r_2 объектов упорядоченного $T \setminus K$;
10: **вернуть** $\mathcal{P}_K(G^+, G^-)$;

5. Добавить предикат B в семейство Z_K.

6. Если $t + 1 \neq t_{max}$, то перейти к следующей итерации. ■

Заметим, что бустинг-алгоритм обучения логического корректора POLAR формирует семейства из предикатов семейства $\mathcal{P}_K(G^+, G^-)$ — подмножества семейства $\mathcal{P}_K$. Это позволяет существенно сократить временные затраты поиска предиката на шаге 3. Напомним, что поиску поляризуемого предиката с максимальной информативностью $I_t(B, K)$ посвящён подраздел 1.3.4.

При выборке семейства $\mathcal{P}_K(G^+, G^-)$ на шаге 2 целесообразно минимизировать число строк матрицы, по которой строятся предикаты из этого семейства. Число строк указанной матрицы равно $|G^+||G^-|$.

На шаге 2 может быть использована процедура `SpecifySearchSpace`, представленная на схеме алгоритма 1. В этой процедуре «жадным» образом определяются параметры семейства $\mathcal{P}_K(G^+, G^-)$, для которого $W_t(G^+, G^-, K) > \delta$ и значение $|G^+||G^-|$ близко к наименьшему.

2.1.3. Сходимость бустинг-алгоритма обучения логического корректора POLAR

Корректность описанного в подразделе 2.1.2 логического корректора POLAR уже не обеспечивается корректностью каждого предиката, участвующего в голосовании. Далее обосновывается, что всегда можно выбрать параметры обучения t_{max} и δ так, чтобы в результате получался корректный распознающий алгоритм.

Лемма 2.1. *Пусть после $t > 0$ итераций построен логический корректор A_t. Если при построении A_t на каждой итерации i, $1 \leqslant i \leqslant t$, в некоторое семейство Z_K добавлялся предикат B с весом α_B, найденным по формуле (2.3), и при этом всякий раз выполнялось неравенство*

$$J^*_{i-1}(B,K) > \delta, \text{ где } \delta = \sqrt{\frac{\ln(m(l-1))}{t}},$$

то распознающий алгоритм A_t корректен.

Доказательство. Подставив (2.3) в (2.1), можно убедиться, что верно неравенство

$$\hat{Q}(A_i) \leqslant \hat{Q}(A_{i-1})\left(1 - (J^*_{i-1}(B,K))^2\right). \tag{2.4}$$

Заметим, что $\hat{Q}(A_0) = m(l-1)$, поскольку $M_0(S_i, K_y) = 0$. Из (2.4) и условия утверждения получаем цепочку неравенств

$$Q(A_t) \leqslant \hat{Q}(A_t) < \hat{Q}(A_0)(1-\delta^2)^t \leqslant m(l-1)e^{-\delta^2 t} = 1.$$

Значение $Q(A_t)$ должно быть целым числом, следовательно $Q(A_t) = 0$. ■

Лемма 2.2. *Если процедура* `SpecifySearchSpace` *запускается с параметром $\delta < \delta^*$, где $\delta^* = \frac{1}{\sqrt{l}} - \frac{1}{\sqrt{2m}}$, то результатом её выполнения является семейство предикатов $\mathcal{P}_K(G^+, G^-)$, для которого $W_t(G^+, G^-, K) > \delta$.*

Доказательство. Заметим, что $W_t(T \cap K, T \setminus K, K) = \sqrt{P_t([S \in T \cap K], K)} - \frac{1}{\sqrt{2m}}$. Нетрудно убедиться, что $P_t([S \in T \cap K_1], K_1) + \ldots + P_t([S \in T \cap K_l], K_l) = 1$. Поэтому всегда найдётся класс K, для которого $P_t([S \in T \cap K], K) \geqslant 1/l$, а следовательно $W_t(T \cap K, T \setminus K, K) \geqslant \delta^*$. То есть, если выполнены условия доказываемого утверждения, то процедура `SpecifySearchSpace` выбирает класс K такой, что $W_t(T \cap K, T \setminus K, K) > \delta$. Наборы G^+ и G^- формируются так, чтобы выполнялось условие $W_t(G^+, G^-, K) > \delta$. Утверждение доказано. ■

Теорема 2.3. *Пусть бустинг-алгоритм обучения логического корректора* RPOLAR *запускается с параметрами*

$$t_{max} > \frac{\ln(m(l-1))}{\delta^2} \text{ и } \delta < \frac{1}{\sqrt{l}} - \frac{1}{\sqrt{2m}}.$$

Тогда в результате его работы строит корректный распознающий алгоритм.

Доказательство. Из леммы 2.2 следует, что для предиката B, строящегося на шаге 3 бустинг-алгоритма обучения логического корректора RPOLAR, верно неравенство $J_{t-1}(B,K) > \sqrt{\ln(m(l-1))/t_{max}}$, $t \in \{1,\ldots,t_{max}\}$. Таким образом, справедливы предпосылки леммы 2.1, из которого заключаем, что распознающий алгоритм $A_{t_{max}}$ корректен. Теорема доказана. ■

2.2. Локальные базисы классов

В данном разделе предлагается поиск голосующих предикатов осуществлять в рамках локальных базисов классов — предварительно формируемых корректных наборов, состоящих из информативных эл.кл. Локальный базис определяет состав столбцов матрицы сравнения, строящейся для поиска поляризуемых предикатов. Разрабатывается алгоритм формирования «хороших» локальных базисов из эл.кл. произвольного ранга. Строится ряд модификаций логического корректора RPOLAR, различающиеся способами формирования локального базиса.

Впервые понятие локального базиса класса применительно к задаче построения логических корректоров было использовано в работах [16, 17]. Локальные базисы позволили снять ограничение на ранг эл.кл. без существенного увеличения временных затрат. В более ранних работах логические корректоры строились с использованием только одноранговых эл.кл. [15, 18]. Фактически набор эл.кл. ранга 1 в указанных работах является локальным базисом, хотя сам термин и не вводился. Позднее плодотворность идеи применения локального базиса класса была подтверждена в [26], где построен логический корректор MONC, строящий локальное базисы простым стохастическим алгоритмом.

Напомним, что через $\mathcal{V}^*$ обозначается множество троек (H,σ,r), где (H,σ) — эл.кл. и r — одно из отношений $[x \leqslant y]$ или $[x \geqslant y]$. Мощность $\mathcal{V}^*$ даже в задаче с небольшим числом признаков может оказаться существенной. Матрица L_T имеет $|\mathcal{V}^*| + |T \cap K|$ столбцов. Большое число столбцов матрицы сравнения затрудняет

поиск предикатов. Предлагается использовать не всю матрицу сравнения, а лишь подматрицу, состоящую из части её столбцов.

Набор $\mathcal{V}_K = \{(H_1, \sigma_1, r_1), \ldots, (H_d, \sigma_d, r_d)\}$ троек из $\mathcal{V}^*$ будем называть *локальным базисом класса* K, если набор эл.кл. $((H_1, \sigma_1), \ldots, (H_d, \sigma_d))$ отделяет прецеденты из $T \cap K$ от прецедентов из $T \setminus K$ с помощью набора отношений $(r_1, \ldots, r_d)$. Ясно, что $\mathcal{V}_K$ является локальным базисом класса K тогда и только тогда, когда подматрица, составленная из столбцов $\mathcal{V}_K$ матрицы L_K, не имеет нулевых строк, то есть для этой подматрицы существует покрытие. Набор $\mathcal{V} \subseteq \mathcal{V}^*$, являющийся локальным базисом для каждого из классов $K_1, \ldots, K_l$ будем называть *локальным базисом задачи*. Например, набор $\mathcal{V}_1$, состоящий из троек $(H, \sigma, r) \in \mathcal{V}^*$ таких, что эл.кл. (H, σ) имеет ранг 1, является локальным базисом задачи.

Опишем универсальный метод построения локального базиса класса, состоящего из эл.кл. произвольного ранга.

Рассмотрим задачу распознавания с двумя классами K и $\overline{K}$. Построим семейство эл.кл. C_K и каждому эл.кл. $(H, \sigma) \in C_K$ присвоим вес $\alpha_{(H,\sigma)} \neq 0$. Рассмотрим распознающий алгоритм

$$A_T^K(S) = \operatorname{sign}\Big(\sum_{(H,\sigma)\in C_K} \alpha_{(H,\sigma)}[H(S) = \sigma]\Big), \tag{2.5}$$

где $\operatorname{sign}(x)$ — функция «знак», возвращающая 1, при $x < 0$, -1, при $x < 0$, и 0, при $x = 0$. Будем считать алгоритм A_T^K корректным в случае, когда $A_T^K(S_i) = 1$, $\forall S_i \in T \cap K$, и $A_T^K(S_i) = -1$, $\forall S_i \in T \setminus K$. Построим по взвешенному семейству C_K набор $\mathcal{V}_K$ такой, что каждому эл.кл. (H, σ) из C_K однозначно соответствует тройка $(H, \sigma, o) \in \mathcal{V}_K$, в которой $o = [x \leqslant y]$, при $\alpha_{(H,\sigma)} > 0$, и $o = [x \geqslant y]$, при $\alpha_{(H,\sigma)} < 0$. Очевидно, что справедливо

Утверждение 2.4. *Если распознающий алгоритм A_T^K корректен, то набор $\mathcal{V}_K$, построенный по взвешенному семейству эл.кл. C_K является локальным базисом класса K. Причём упорядоченный набор, составленный из эл.кл. семейства C_K, является корректным для K и имеет поляризуемую корректирующую функцию.*

Существует ряд методов построения корректных распознающих алгоритмов вида (2.5), например, бустинг или построение деревьев решений. В [16] лучшим алгоритмом построения локального базиса оказался бустинг-алгоритм BOOSTLO. В

настоящей работе используется два метода: голосование по представительным наборам и бустинг-алгоритм, аналогичный BOOSTLO.

Практика показывает, что для прикладной задачи с большой значностью признаков редко удаётся построить небольшой локальный базис. Заметим, что при использовании бустинга для формирования семейств голосующих предикатов на каждой итерации ищется набор эл.кл. U, отделяющий некоторое подмножество прецедентов G от подмножества прецедентов G^-. При этом совсем не обязательно осуществлять поиск набора U в локальном базисе задачи. Целесообразно на каждой итерации формировать локальный базис, ориентированный на отделение G от G^- и учитывающий текущие веса остальных прецедентов.

2.3. Реализация и экспериментальное исследование логических корректоров POLAR

Были реализованы и протестированы 4 модификации логического корректора POLAR, отличающиеся стратегией формирования локального базиса. Каждая из модификаций для формирования семейств голосующих предикатов использует бустинг.

1. POLAR-1 — логический корректор, в котором предикаты строятся в рамках локального базиса, состоящего из троек (H, σ, r) таких, что ранг эл.кл. (H, σ) равен 1 и отношение $r \in \{[x \leqslant y], [x \geqslant y]\}$.

2. POLAR-2 — логический корректор, в котором предикаты строятся в рамках локального базиса задачи, построенного бустинг-алгоритмом.

3. POLAR-3 — логический корректор, в котором предикаты строятся в рамках локального базиса, формируемого на каждой итерации голосованием по представительным наборам.

4. POLAR-4 — логический корректор, в котором предикаты строятся в рамках локального базиса, формируемого на каждой итерации бустинг-алгоритмом.

Новые логические корректоры были протестированы на прикладных задачах из репозитория UCI. В таблице 2.1 даны характеристики задач. В столбцах l, m, n

Таблица 2.1: Задачи распознавания

№	Название	l	m	n	z
1.	audiology	24	226	69	161
2.	balance scale	3	625	4	20
3.	breast cancer	2	699	9	90
4.	car	4	1728	6	21
5.	dermatology	4	366	34	192
6.	house votes	2	435	16	48
7.	kr vs kp	2	3196	36	73
8.	monks-2	2	601	6	17
9.	nursery	5	12960	8	27
10.	soybean large	19	307	35	132
11.	tic tac toe	2	958	9	27
12.	optdigits	10	5620	64	914
13.	letter recognition	26	20000	16	256
14.	lenses	3	24	4	9
15.	soybean small	4	47	35	72

и z приведены соответственно число классов, число строк, число столбцов и число всех представительных наборов ранга 1, характеризующее значность признаков.

Задачи по трудоёмкости можно разбить на 3 группы. Задачи с номерами 1 – 11 имеют средний объём обучения, и поэтому для тестирования на этих задачах применяется методика 10-кратного скользящего контроля. В задачах 12 и 13 много объектов, поэтому для сокращения времени счёта выборка делится только 1 раз на обучающую и тестовую. В задачах 14 и 15 наоборот очень мало объектов, поэтому используется методика скользящего контроля по одному объекту (leave-one-out).

В тестировании помимо логических корректоров POLAR-1 – POLAR-4 участвовали следующие алгоритмы распознавания:

1) Т — голосование по тестам (для каждого класса строится не более 200 тестов);

2) ПН — голосование по представительным наборам (для каждого объекта строится не более 5 представительных наборов);

3) МОН — голосование по монотонным корректным наборам эл.кл. (для каждого класса строится не более 200 наборов и эл.кл. имеют ранг 1);

4) Т* — голосование по тестам (голосующие семейства формируются бустингом);

Таблица 2.2: Средняя частота ошибок на тестовой выборке

		Классические			Бустинг			Лог. кор. POLAR			
№	Задача	T	ПН	МОН	T*	ПН*	МОН*	1	2	3	4
1.	audiology	0.14	0.07	0.09	0.03	0.03	0.03	0.03	0.03	**0.02**	0.03
2.	b. scale	0.92	0.27	0.46	0.25	0.2	0.19	**0.18**	0.23	0.23	0.25
3.	b. cancer	0.21	0.05	0.24	0.046	**0.044**	0.057	0.061	0.059	0.065	0.059
4.	car	0.97	0.09	0.27	0.061	0.032	0.033	0.013	0.027	0.022	**0.011**
5.	dermat.	0.84	0.47	0.79	0.41	0.4	0.4	**0.39**	0.42	0.44	0.43
6.	h. votes	0.34	0.06	0.15	0.07	**0.05**	0.07	**0.05**	0.06	0.07	0.08
7.	kr-vs-kp	0.63	0.017	0.101	0.008	0.004	**0.003**	0.008	0.007	0.004	**0.003**
8.	monks-2	0.96	0.52	0.96	0.37	0.55	0.42	**0.04**	0.44	0.56	0.36
9.	nursery	0.66	0.015	0.36	0.027	0.003	0.005	0.002	—	**0.0019**	0.004
10.	soybean l.	0.19	0.094	0.131	0.075	**0.064**	0.072	0.078	0.106	0.083	0.075
11.	tic-tac-toe	0.97	0.005	0.52	0.011	0.002	0.005	0.028	0.002	**0.001**	0.007
12.	letter r.	0.52	0.21	0.63	0.21	**0.16**	0.25	—	—	0.23	0.25
13.	optdigits	0.77	0.19	0.55	0.25	0.23	0.17	0.15	—	0.27	**0.14**
14.	lenses	1	**0.21**	0.46	0.42	0.25	0.29	0.33	0.29	0.38	0.25
15.	soybean s.	0.02	**0**	**0**	**0**	**0**	**0**	**0**	0.02	0.04	**0**

5) ПН* — голосование по представительным наборам (голосующие семейства формируются бустингом);

6) МОН* — голосование по монотонным корректным наборам эл.кл. (голосующие семейства формируются бустингом и эл.кл. в наборах имеют ранг 1).

В таблице 2.2 приведены результаты счёта. Показателем качества является средняя доля ошибок на тестовых выборках. Прочерки соответствуют случаям, когда алгоритм не справился с задачей за 1 час.

На 14 задачах лидируют алгоритмы, в которых применяется бустинг для формирования голосующих семейств. На 11 задачах лидируют новые модели. Лучшими среди новых являются POLAR-3 и POLAR-4, в которых локальный базис формируется на каждой итерации. Причём эти логические корректоры демонстрируют хорошие результаты на задачах с большой значностью признаков и имею сравнительно небольшое время счёта почти на всех задачах.

Время счёта представлено в таблице 2.3. Жирным шрифтом выделено время работы самого быстрого алгоритмами для каждой из задач. Почти всегда это классическое голосование по представительным наборам. Наименьшее время работы логи-

Таблица 2.3: Время счёта в секундах

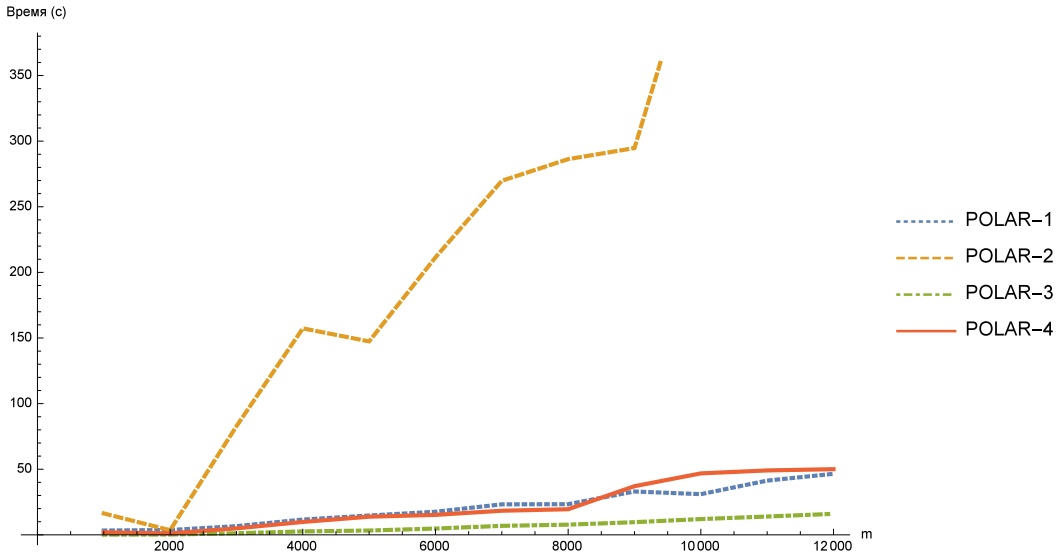
		Классические			Бустинг			Лог. кор. POLAR			
№	Задача	T	ПН	МОН	T*	ПН*	МОН*	1	2	3	4
1.	audiology	1.9	**1.4**	4.6	22.7	8.9	42.2	224.1	420.2	4.1	82.4
2.	b. scale	0.6	**0.4**	2.4	0.8	1.1	2.8	132.5	1251.1	3.9	243.7
3.	b. cancer	1.2	**0.2**	7.8	0.7	0.5	5.2	108.1	110.1	1.3	51.3
4.	car	3.1	**1.3**	10.1	2.3	3.1	7.1	78.5	713.7	7.9	34.9
5.	dermat.	**2.8**	15.4	13.2	40.9	66.5	118.9	272.4	689.6	98.9	345.1
6.	h. votes	2.4	**1.1**	7.6	4.2	8.6	12.1	37.1	87.3	7.9	167.3
7.	kr-vs-kp	36.3	**10.2**	94.4	58.9	79.8	87.5	226.1	192.1	84.8	173.6
8.	monks-2	**0.5**	0.6	1.2	0.9	1.9	2.1	15.4	500.6	5.8	104.1
9.	nursery	163.9	**20.9**	595.5	87.9	89.3	224.4	452.9	—	157.9	589.1
10.	soybean l.	2.5	**2.4**	7.7	28.3	21.2	79.9	329.3	868.6	15.9	249.2
11.	tic-tac-toe	3.2	**0.6**	6.5	4.5	1.9	10.2	45.6	9.3	2.2	16.2
12.	letter r.	58.2	**47.1**	790.7	92.3	233.1	550.5	—	—	363.1	1191.1
13.	optdigits	**25.8**	636.2	277.7	249.6	1570.5	840.6	3117.2	—	2160.6	1110.8
14.	lenses	**0.01**	**0.01**	0.03	**0.01**	0.03	0.06	0.09	4.7	0.03	2.2
15.	soybean s.	0.4	**0.06**	1.1	0.8	0.1	1.5	4.8	0.3	0.1	0.4

ческих корректоров на соответствующей задаче подчёркнуто. Самым быстрым всегда оказывается либо МОН, либо POLAR-3.

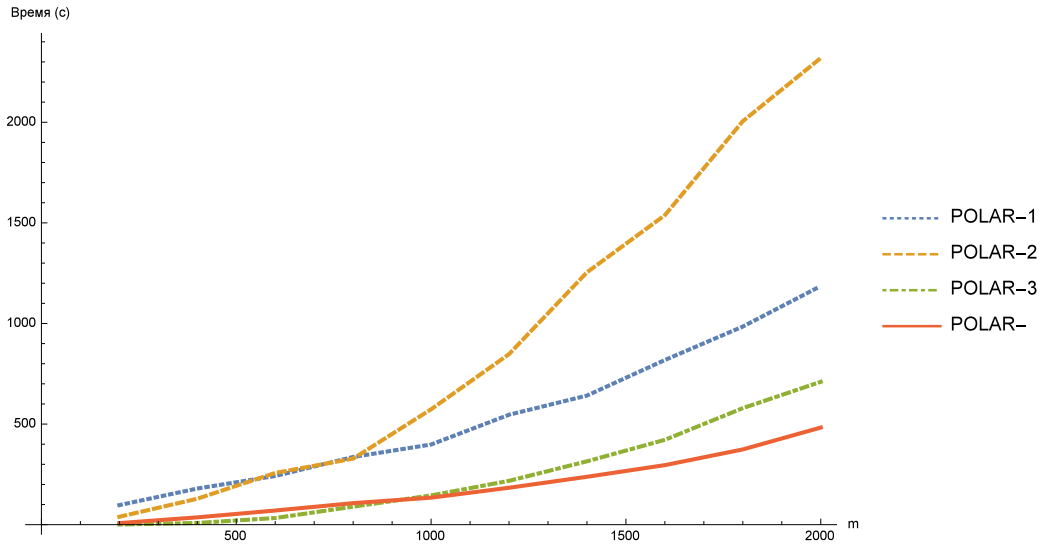
Для выявления наилучшей стратегии построения локального базиса с точки зрения времени счёта проведена следующая серия экспериментов. Выбраны две задачи с достаточно большим числом объектов: nursery и optdigits. Задача nursery отличается от задачи optdigits тем, что имеет сравнительно небольшую значность и существенно неравномерное распределение объектов по классам. Для задачи nursery было сформировано 60 случайных подвыборок по 5 подвыборок каждого из размеров $1000, 2000, \ldots, 12000$. Для задачи optdigits было сформировано 50 подвыборок по 5 подвыборок каждого из размеров $200, 400, \ldots, 2000$. На рисунках 1.1а и 1.1б изображены графики зависимости усреднённого времени счёта POLAR-1–POLAR-4 от размера подвыборки.

Очевидно, самой неудачной является модификация POLAR-2. Время работы POLAR-2 быстро увеличивается с ростом объёма обучения, напрямую связанного с мощностью строящегося логическим корректором локального базиса задачи.

На задаче nursery POLAR-3 является наилучшим. Строящиеся логическим корректором POLAR-3 локальные базисы имеют небольшую мощность, поскольку в



(a) nursery



(b) optdigits

Рисунок 2.1: Зависимость среднего времени обучения логических корректоров POLAR-1–POLAR-4 от размера выборки

задачах с малой значностью признаков, как правило, представительные наборы имеют высокую информативность.

На задаче optdigits POLAR-4 обгоняет POLAR-3, начиная с размера подвыборки 800. Бустинг-алгоритм, использующийся в POLAR-4 для формирования локального базиса не требует корректности эл.кл. Большая значность признаков в задаче optdigits приводит к тому, что в локальный базис, формируемый POLAR-3, попадает много малоинформативных представительных наборов, что плохо сказывается на времени счёта.

ГЛАВА 3

Новые асимптотически оптимальные алгоритмы дуализации

В данной главе рассматривается одна из центральных дискретных перечислительных задач —дуализация. Даётся обзор основных подходов её решения, среди которых выделяется подход к построению асимптотически оптимальных алгоритмов. Алгоритмы, построенные в рамках этого подхода классифицируются на два типа. Строятся новые асимптотически оптимальные алгоритмы первого типа АО1К, АО1М, АО2К и АО2М, и второго типа RUNC, RUNC-M и PUNC. Новые и ранее построенные асимптотически оптимальные алгоритмы дуализации экспериментально исследуются на большом объеме разнотипных данных.

3.1. Задача дуализации и подходы к её решению

Пусть $L = \|a_{ij}\|_{m\times n}$ — булевая матрица и пусть H — набор столбцов матрицы L. Набор H называется *покрытием* матрицы L, если каждая строка матрицы L в пересечении хотя бы с одним столбцом из H дает 1. Покрытие H называется *неприводимым*, если любое собственное подмножество H не является покрытием L. Обозначим через $\mathcal{P}(L)$ множество всевозможных неприводимых покрытий матрицы L. Требуется построить множество $\mathcal{P}(L)$.

В теории алгоритмической сложности задача перечисления элементов $\mathcal{P}(L)$ считается главной задачей перечисления и называется дуализацией. Существуют другие формулировки этой задачи, в частности с использованием понятий теории булевых функций и теории графов и гиперграфов. Приведем эти формулировки.

1. Дана конъюнктивная нормальная форма из m различных элементарных дизъюнкций, реализующая монотонную булеву функцию $F(x_1, \ldots, x_n)$. Требуется построить сокращённую дизъюнктивную нормальную форму функции F.

2. Дан гиперграф $\mathcal{H}$ с n вершинами и m ребрами. Требуется найти все минимальные вершинные покрытия гиперграфа $\mathcal{H}$.

3.1.1. Подходы к оценке сложности алгоритмов дуализации

Эффективность алгоритмов перечисления принято оценивать сложностью шага [27]. Говорят, что алгоритм работает с (квази)полиномиальной задержкой, если для любой индивидуальной задачи каждый шаг алгоритма (построение очередного решения) осуществляется за (квази)полиномиальное время от размера задачи. В применении к поиску неприводимых покрытий это означает, что для любой булевой матрицы размера $m \times n$ время построения очередного неприводимого покрытия должно быть ограничено полиномом или квазиполиномом от m и n. В общем случае алгоритм дуализации с (квази)полиномиальной задержкой до сих пор не построен и неизвестно, существует ли он. Известны примеры таких алгоритмов для некоторых частных случаев дуализации [27, 71–73]. Например, в [27] построен алгоритм с задержкой $\mathcal{O}(n^3)$ для случая, когда в каждой строке матрицы L не более двух единичных элементов, что в постановке 2) соответствует случаю: $\mathcal{H}$ — граф. Также встречаются работы, в которых доказывается, что алгоритм, имеющий определённую конструкцию, не может в худшем случае работать с полиномиальной задержкой (например см. [74]).

Исследования в области сложности перечислительных задач в основном касаются изучения возможности построения инкрементальных (квази)полиномиальных алгоритмов. В данном случае инкрементальность означает, что алгоритму разрешено на каждом шаге (при построении очередного решения) просматривать множество решений, построенных на предыдущих шагах, и на этот просмотр тратить время, (квази)полиномиальное от размера задачи и числа уже найденных решений. В [28, 31] построен инкрементальный квазиполиномиальный алгоритм дуализации. В [29, 30] для ряда частных случаев дуализации построены инкрементальные полиномиальные алгоритмы.

3.1.2. Асимптотически оптимальный подход

Существует еще один подход к решению рассматриваемой задачи, основанный на понятии асимптотически оптимального алгоритма с полиномиальной задержкой. Подход впервые предложен в [35] и ориентирован на типичный случай (on average). При определенных условиях этот подход позволяет заменить исходную перечислительную задачу Z на более «простую» перечислительную задачу Z_1, имеющую тот же вход и решаемую с полиномиальной задержкой. При этом, во-первых, множество решений задачи Z_1 содержит множество решений задачи Z, и во-вторых, почти всегда с ростом размера входа число решений задачи Z_1 асимптотически равно числу решений задачи Z. Обоснование данного подхода базируется на получении асимптотик для типичного числа решений каждой из задач Z и Z_1.

Таким образом, в отличие от «точного» алгоритма с полиномиальной задержкой асимптотически оптимальному алгоритму разрешено делать «лишние» полиномиальные шаги. Лишний шаг — это построение такого решения задачи Z_1, которое либо было найдено ранее, либо построено впервые, но не является решением задачи Z. Число лишних шагов для почти всех задач данного размера должно иметь более низкий порядок роста, чем число всех шагов алгоритма, с ростом размера задачи. Проверка того, является ли шаг лишним должна осуществляться за полиномиальное время от размера задачи.

К настоящему моменту для случая, когда $\log m \leqslant (1-\varepsilon)\log n$, $0 < \varepsilon < 1$, построен ряд асимптотически оптимальных алгоритмов поиска неприводимых покрытий булевой матрицы [32–40]. В этих алгоритмах для перечисления $\mathcal{P}(L)$ используется следующий критерий.

Набор H из r столбцов матрицы L является неприводимым покрытием тогда и только тогда, когда выполнены следующие два условия: 1) подматрица L^H матрицы L, образованная столбцами набора H, не содержит строки вида $(0,0,\ldots,0)$; 2) L^H содержит каждую из строк вида $(1,0,0,\ldots,0,0),(0,1,0,\ldots,0,0),\ldots,(0,0,0,\ldots,0,1)$, то есть с точностью до перестановки строк содержит единичную подматрицу порядка r. Набор столбцов, удовлетворяющий условию 2), называется *совместимым*.

В асимптотически оптимальном алгоритме дуализации АО1 [35] в роли Z_1 выступает задача построения совокупности наборов столбцов матрицы L, удовлетво-

ряющих условию 2), в которой каждый набор длины r встречается столько раз, сколько единичных подматриц порядка r этот набор содержит. Фактически с полиномиальной задержкой перечисляются все единичные подматрицы матрицы L. Ясно, что неприводимое покрытие может порождаться только максимальной единичной подматрицей, то есть такой единичной подматрицей, которая не содержится в других единичных подматрицах. Максимальная единичная подматрица порождает максимальный совместимый набор столбцов, то есть такой совместимый набор столбцов, который не содержится ни в каком другом совместимом наборе столбцов.

Схема работы алгоритма АО1 позволяет со сложностью шага $\mathcal{O}(qmn)$, где $q = \min\{m, n\}$, перечислять максимальные единичные подматрицы (перечислять с повторениями максимальные совместимые наборы столбцов). Перебор единичных подматриц приводит к тому, что некоторые наборы столбцов строятся неоднократно. При получении очередной максимальной единичной подматрицы Q за время $\mathcal{O}(mn)$ алгоритм АО1 проверяет условие 1) для набора столбцов H матрицы L, который порождается подматрицей Q, и, если условие 1) выполнено, то алгоритм АО1 за время $\mathcal{O}(mn)$ проверяет не был ли набор H построен на предыдущих шагах.

В алгоритме АО2 [37], который является модификацией алгоритма АО1, с полиномиальной задержкой $\mathcal{O}(qm^2n)$ перечисляются только такие единичные подматрицы матрицы L, которые порождают покрытия. Алгоритм АО2 строит на каждом шаге неприводимое покрытие, однако найденные решения также, как и в алгоритме АО1, могут повторяться. Этот алгоритм делает меньше лишних шагов по сравнению с алгоритмом АО1. В [55] на базе алгоритма АО2 были построены алгоритмы АО2К и АО2М, позволяющие сократить время счёта.

Наименьшее число лишних шагов имеет асимптотически оптимальный алгоритм ОПТ [39], основанный на перечислении с полиномиальной задержкой $\mathcal{O}(qm^2n)$ наборов столбцов матрицы L, удовлетворяющих условию 2) и некоторым дополнительным условиям, среди которых условие максимальности. Лишние шаги в алгоритме ОПТ возникают за счёт построения максимальных совместимых наборов столбцов, не являющихся покрытиями, то есть не удовлетворяющих условию 1).

Пусть $\mathcal{S}(L)$ – множество всех единичных подматриц матрицы L. Как уже отмечалось, перечисление элементов $\mathcal{S}(L)$ может быть осуществлено с полиномиальной задержкой $\mathcal{O}(mn)$.

Обоснование асимптотически оптимальных алгоритмов дуализации базируется на следующем утверждении. Если $\log m \leqslant (1-\varepsilon)\log n$, $0 < \varepsilon < 1$, то для любого сколь угодно малого $\delta > 0$ существует предел

$$\lim_{m,n\to\infty} \mathsf{P}\{\frac{|\mathcal{S}(L)|}{|\mathcal{P}(L)|} < 1+\delta\} = 1,$$

где вероятность под знаком предела вычисляется при условии, что каждая матрица L фиксированного размера $m \times n$ выбирается с вероятностью 2^{-mn}. Доказательство этого утверждения и ряда более сильных утверждений можно найти в [75].

В [76, 77] предложены алгоритмы дуализации RS и MMCS, для описания которых используются понятия теории гиперграфов. Эти алгоритмы основаны на построении наборов вершин S гиперграфа $\mathcal{F}$, удовлетворяющих условию «crit»: $crit(v,S) \neq \varnothing, \forall v \in S$. Условие «crit» эквивалентно условию «совместимости» 2) для соответствующего набора столбцов матрицы инцидентности гиперграфа $\mathcal{F}$. Таким образом, предложенный в [76, 77] подход к построению алгоритмов дуализации не является новым (алгоритмы RS и MMCS фактически являются асимптотически оптимальными). В [76, 77] показано, что алгоритмы RS и MMCS по скорости значительно превосходят ряд других алгоритмов [28, 78–81].

3.1.3. Основные типы асимптотически оптимальных алгоритмов дуализации

Асимптотически оптимальные алгоритмы дуализации можно условно разделить на два типа. К первому типу относятся алгоритмы, перечисляющие с полиномиальной задержкой максимальные единичные подматрицы матрицы L. Такие алгоритмы совершают лишние шаги, связанные с повторным построением максимальных совместимых наборов столбцов. Алгоритмы второго типа основаны на перечислении с полиномиальной задержкой без повторений максимальных совместимых наборов столбцов.

Примерами алгоритмов первого типа служат алгоритмы АО1 [35] и АО2 [37]. В разделе 3.2 построены модификации АО1К, АО1М алгоритма АО1, и описаны модификации АО2К, АО2М алгоритма АО2, построенные в [55], проанализирована сложность шага, и проведено экспериментальное исследование алгоритмов первого типа. Показано, что построенные модификации алгоритмов АО1 и АО2, как правило, работают быстрее своих прототипов.

Примерами алгоритмов второго типа являются алгоритмы ОПТ [39], MMCS, RS [76, 77], и разработанные автором алгоритмы RUNC, RUNC-M, PUNC [53, 57]. В разделе 3.3 дается общая схема алгоритма второго типа, в рамках схемы описывается каждый из указанных алгоритмов. Далее даёт детальное описание алгоритмов RUNC, RUNC-M и PUNC, анализируется сложность их шага, и осуществляется экспериментальное исследование алгоритмов второго типа. Показано, что алгоритмы, построенные в [53, 57], требуют меньших временных затрат по сравнению с асимптотически оптимальными алгоритмами, построенными ранее в [32–40, 55] и в большинстве случаев опережают алгоритмы из [76, 77].

Оригинальные описания алгоритмов АО1, АО2, АО2К, АО2М, ОПТ, RS и MMCS отличаются от приведённых в настоящей работе, в связи со стремлением автора использовать общие понятия и обозначения для описания асимптотически оптимальных алгоритмов первого и второго типов.

3.2. Асимптотически оптимальные алгоритмы дуализации первого типа

Приводится схема работы асимптотически оптимального алгоритма первого типа. В рамках этой схемы описываются алгоритмы АО1, АО2 и строятся их модификации АО1К, АО1М, АО2К, АО2М, в которых сокращаются вычислительные затраты за счёт уменьшения общего числа вершин дерева решений.

3.2.1. Общая схема алгоритма дуализации первого типа

Введем используемые далее понятия и обозначения. Будем говорить, что столбец j (столбец с номером j) *покрывает* строку i (строку с номером i) матрицы L, если $a_{ij} = 1$.

Пусть H — набор столбцов матрицы L. Будем говорить, что набор H покрывает строку i, если существует столбец $j \in H$, покрывающий строку i.

Обозначим через $E(L)$ множество $\{(i,j) : a_{ij} = 1, i \in \{1,\ldots,m\}, j \in \{1,\ldots,n\}\}$. Два элемента (i,j) и (t,l) из $E(L)$ будем называть *совместимыми*, если $a_{il} = 0,\ a_{tj} = 0$. Набор Q элементов из $E(L)$ будем называть *совместимым*, если любые два различных элемента (i,j) и (t,l) из Q совместимы. Совместимый

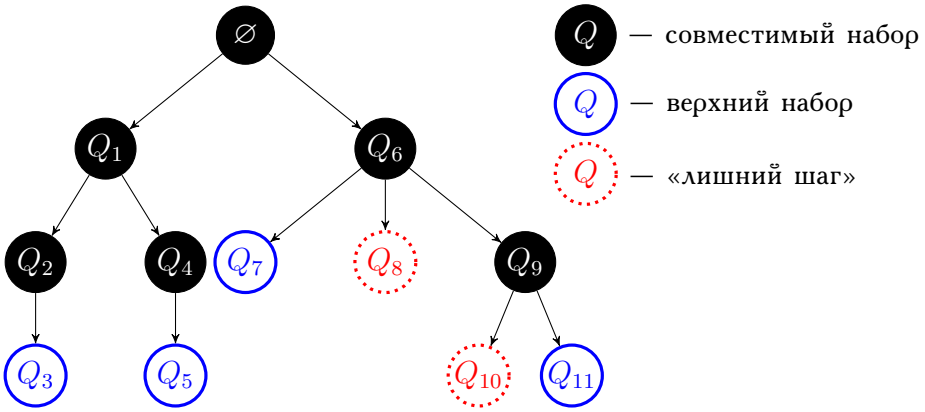


Рисунок 3.1: Дерево решений асимптотически оптимального алгоритма дуализации первого типа

набор Q будем называть *максимальным*, если Q не является подмножеством другого совместимого набора элементов из $E(L)$.

Пусть Q — совместимый набор элементов из $E(L)$. Столбец l называть *запрещённым* для Q, если существует элемент $(i,j) \in Q$ такой, что столбец l покрывает строку i. В противном случае будем говорить, что столбец l *совместим* с набором Q.

Пусть $B \subseteq E(L)$. Обозначим набор столбцов $\{j\colon \exists (i,j) \in B\}$ через $H(B)$. Будем говорить, что набор B элементов из $E(L)$ *порождает* набор столбцов $H(B)$. Будем говорить, что строка i матрицы L *покрыта* набором B элементов из $E(L)$, если набор столбцом $H(B)$ покрывает строку i. Совместимый набор Q назовем *покрывающим*, если все строки матрицы L покрыты набором Q. Очевидно, что покрывающий набор является максимальным, и справедливо

Утверждение 3.1. *Набор столбцов H является неприводимым покрытием матрицы L тогда и только тогда, когда найдется покрывающий набор Q, для которого $H(Q)=H$.*

Покрывающий набор $Q = \{(i_1, j_1), \ldots, (i_r, j_r)\}$ называется *верхним*, если для любого покрывающего набора $Q' = \{(t_1, j_1), \ldots, (t_r, j_r)\}$ верны неравенства $t_u \geqslant i_u$, $u \in \{1, \ldots, r\}$. Очевидно

Утверждение 3.2. *Для любого неприводимого покрытия H существует единственный верхний набор Q, такой, что $H(Q) = H$.*

Таким образом, из утверждений 3.1 и 3.2 следует, что задача построения $\mathcal{P}(L)$ сводится к перечислению верхних наборов элементов из $E(L)$.

Работу асимптотически оптимальных алгоритмов дуализации первого типа можно представить в виде одностороннего обхода ветвей дерева решений, вершины которого, за исключением корня, — совместимые наборы элементов из $E(L)$. Корень дерева — пустой набор. Висячие вершины либо являются верхними наборами, либо соответствуют лишним шагам алгоритма. Схема дерева решений изображена на рис. 3.1.

Каждый шаг алгоритма является итеративной процедурой, в результате которой строится одна ветвь дерева, начинающаяся либо в корне, либо в некоторой построенной ранее внутренней вершине. Например, алгоритм, строящий дерево на рис. 3.1, выполняется 6 шагов:

1) $\varnothing \to Q_1 \to Q_2 \to Q_3$;

2) $\varnothing \to Q_1 \to Q_4 \to Q_5$;

3) $\varnothing \to Q_6 \to Q_7$;

4) $Q_6 \to Q_8$ (лишний шаг);

5) $Q_6 \to Q_9 \to Q_{10}$ (лишний шаг);

6) $Q_9 \to Q_{11}$.

При переходе от вершины к вершине меняется состояние алгоритма. Для обозначения того, что некоторый объект X, описывающий состояние алгоритма, связан с вершиной Q будем писать $X[Q]$.

Общая схема асимптотически оптимального алгоритма дуализации первого типа. На шаге 1 на итерации 1 по некоторому правилу формируется набор $B[\varnothing]$ элементов из $E(L)$, корень становится текущей вершиной, и происходит переход к итерации 2.

Пусть на шаге $s, s \geqslant 1$, на итерации $t, t \geqslant 1$, текущей стала вершина Q. Тогда на итерации $t+1$ выполняется следующее.

1. Если $B[Q] = \varnothing$, то происходит переход к следующему шагу (в случае, когда Q является висячей вершиной, шаг алгоритма считается лишним). В противном случае берётся некоторый элемент $(i, j) \in B[Q]$.

2. Выбранный элемент (i, j) удаляется из $B[Q]$, и строится вершина $Q' = Q \cup \{(i, j)\}$.

3. Если набор Q' является верхним, то набор Q' становится результатом шага (набор столбцов $H(Q')$ — неприводимое покрытие), и происходит переход к следующему шагу. В противном случае по некоторому правилу строится набор $B[Q']$ элементов из $B[Q]$ такой, что каждый элемент из $B[Q']$ совместим с элементом (i, j).

4. Текущей вершиной становится Q', и происходит переход к следующей итерации.

Пусть результатом шага $s, s \geqslant 1$, является набор Q. Тогда на шаге $s + 1$ на итерации 1 среди вершин ветки дерева, соединяющей корень с вершиной Q, ищется ближайшая к Q вершина Q' такая, что $B[Q'] \neq \varnothing$. Если вершина Q' найдена, то она становится текущей вершиной, и происходит переход к следующей итерации. В противном случае алгоритм завершает работу. ■

Алгоритмы второго типа различаются правилами построения набора $B[\varnothing]$ на шаге 1 на итерации 1 и построения набора $B[Q']$ при создании новой вершины Q'. Опишем эти различия на примере алгоритмов АО1 и АО2.

3.2.2. Алгоритм АО1

Введем дополнительные обозначения. Пусть R — набор строк и C — набор столбцов матрицы L. Обозначим через $L(R, C)$ подматрицу, образованную строками из R и столбцами из C, и через $E(L(R, C))$ множество $\{(i, j) : a_{ij} = 1, i \in R, j \in C\}$.

Алгоритм АО1. На шаге 1 на итерации 1 строится подматрица $L(R[\varnothing], C[\varnothing])$, состоящая из всех строк и столбцов матрицы L, строится набор $B[\varnothing] = E(L)$, корень становится текущей вершиной, и происходит переход к следующей итерации.

Пусть на шаге $s, s \geqslant 1$, на итерации $t, t \geqslant 1$, текущей стала вершина Q. Тогда на итерации $t + 1$ выполняется следующее.

1. Если $B[Q] = \varnothing$, то происходит переход к следующему шагу. В противном случае берется элемент $(i, j) \in B[Q]$ с наименьшим значением $jm + i$.

2. Выбранный элемент (i, j) удаляется из $B[Q]$. Столбцы, не входящие в набор $H(B[Q])$, удаляются из подматрицы $L(R[Q], C[Q])$. Строится вершина $Q' = Q \cup \{(i, j)\}$.

3. Если набор Q' является верхним, то набор Q' становится результатом шага (набор столбцов $H(Q')$ — неприводимое покрытие), и происходит переход к

следующему шагу. В противном случае строится подматрица $L(R[Q'], C[Q'])$ путем удаления из подматрицы $L(R[Q], C[Q])$ строк, покрытых столбцом j, и столбцов, покрывающих строку i.

4. Строится множество $B[Q'] = E(L(R[Q'], C[Q']))$.

5. Текущей вершиной становится Q', и происходит переход к следующей итерации.

На шаге $s, s > 1$, на итерации 1 алгоритм АО1 выполняет те же действия, которые описаны в общей схеме работы алгоритма первого типа. ■

Вершины дерева решений алгоритма АО1, соответствующие лишним шагам, являются максимальными совместимыми наборами элементов из $E(L)$ (подматрица $L(R[Q], C[Q])$ не содержит ни одного единичного элемента). При этом шаг, заканчивающийся в вершине Q, оказывается лишним по одной из двух причин: либо Q не является покрывающим набором, либо Q не является верхним набором.

При реализации алгоритма АО1 нет необходимости явно строить множество $B[Q]$, т.к. элементы этого множества однозначно соответствуют единичным элементам подматрицы $L(R[Q], C[Q])$.

3.2.3. Алгоритм АО2

Для описания алгоритма АО2 потребуются дополнительные понятия и обозначения. Будем говорить, что строка $i \in R$ является *охватывающей* для строки $t \in R$ в подматрице $L(R, C)$, если $a_{ij} \geqslant a_{tj}$, $\forall j \in C$. Легко показать, что если из подматрицы $L(R, C)$ удалить строку $i \in R$, охватывающую строку $t \in R$, то множества неприводимых покрытий полученной и исходной подматриц будут совпадать.

Строку $i \in R$, охватывающую строку $t \in R$ в подматрице $L(R, C)$, будем называть *охватывающей снизу*, если $i > t$. Легко показать, что если из подматрицы $L(R, C)$ удалить строку $i \in R$, охватывающую снизу строку $t \in R$, то множества верхних наборов единичных элементов полученной и исходной подматриц будут совпадать.

Пусть в подматрице $L(R, C)$ строка $i \in R$ охватывает строку $t \in R$, и столбец $j \in C$ покрывает строку i и не покрывает строку t. Элемент (i, j) будем называть *запрещённым* в подматрице $L(R, C)$. Очевидно, если элемент (i, j) является запрещённым в матрице L, то ни один покрывающий набор Q не содержит элемент (i, j).

Алгоритм AO2. На шаге 1 на итерации 1 строится подматрица $L(R[\varnothing], C[\varnothing])$ путем последовательного удаления из матрицы L охватывающих снизу строк матрицы L. Далее с корнем связывается набор $B[\varnothing]$ элементов из $E(L(R[\varnothing], C[\varnothing]))$, незапрещённых в подматрице $L(R[\varnothing], C[\varnothing])$. Корень становится текущей вершиной, и происходит переход к следующей итерации.

Пусть на шаге $s, s \geqslant 1$, на итерации $t, t \geqslant 1$, текущей стала вершина Q. Тогда на итерации $t+1$ выполняется следующее.

1–3. Выполняются пункты 1–3 итерации $t+1$ алгоритма AO1.

4. Из подматрицы $L(R[Q'], C[Q'])$ последовательно удаляются охватывающие снизу строки.

5. Строится набор $B[Q']$ элементов из $E(L(R[Q'], C[Q']))$, незапрещённых в подматрице $L(R[Q'], C[Q'])$.

6. Текущей вершиной становится Q', и происходит переход к следующей итерации.

На шаге $s, s > 1$, на итерации 1 алгоритм AO2 выполняет те же действия, которые описаны в общей схеме работы алгоритма первого типа. ■

Вершины дерева решений алгоритма AO2, соответствующие лишним шагам, являются покрывающими и не являются верхними наборами элементов из $Q(L)$. Таким образом, каждый шаг алгоритма AO2 заканчивается в вершине Q такой, что набор столбцов $H(Q)$ — неприводимое покрытие.

При реализации алгоритма AO2 нет необходимости явно строить множество $B[Q]$. Вместо этого эффективнее работать с набором элементов, запрещённых в подматрице $L(R[Q], C[Q])$.

3.2.4. Сложность задачи идентификации лишнего шага

При описании схемы работы алгоритмов AO1 и AO2 были отмечены причины, по которым алгоритмы делают лишние шаги, и указаны свойства «лишних» висячих вершин дерева решений. Возникает вопрос, существует ли асимптотически оптимальный алгоритм первого типа, перечисляющий неприводимые покрытия без повторений с полиномиальной задержкой, то есть не делающий лишних шагов. Из доказанной ниже теоремы 3.3 следует, что при $\mathsf{P} \neq \mathsf{NP}$ ответ на этот вопрос отрицателен. Аналогичная техника доказательства используется в [29, Proposition 2].

Задача 3.1. *Вход:* L — булева матрица, Q — совместимый набор элементов из $E(L)$. *Выход:* 1, если существует верхний набор $Q' \colon Q \subseteq Q'$, и 0 — иначе.

Теорема 3.3. *Задача 3.1* NP*-полна.*

Доказательство. Пусть булева функция $f(x_1, \ldots, x_N)$ задана конъюнктивной нормальной формой (КНФ) $C_1 \wedge \ldots \wedge C_M$, где $C_1, \ldots, C_M$ — элементарные дизъюнкции от переменных $x_1, \ldots, x_N$.

Построим граф $G = (V, E)$, где множество V состоит из вершин $1, \varnothing, C_1, \ldots, C_M$, $x_1, \ldots, x_N, \bar{x}_1, \ldots, \bar{x}_N$, и множество E состоит из ребер четырех типов:

1) ребра для конъюнкций $\{1, C_1\}, \ldots, \{1, C_M\}$;

2) вспомогательное ребро $\{1, \varnothing\}$;

3) ребра для переменных $\{x_1, \bar{x}_1\}, \ldots, \{x_N, \bar{x}_N\}$;

4) ребра для вхождений переменных в конъюнкции: $\{C_i, x_j\}$, если x_j входит в C_i, и $\{C_i, \bar{x}_j\}$, если $\bar{x}_j$ входит в C_i.

Пусть L — матрица инцидентности графа G и столбцы с номерами $1, \ldots, M+2$ соответствуют вершинам $1, \varnothing, C_1, \ldots, C_M$, а строки с номерами $1, \ldots, M, M+1$ — ребрам $\{1, C_1\}, \ldots, \{1, C_M\}, \{1, \varnothing\}$.

Рассмотрим совместимый набор $Q = \{(M+1, 1)\}$. Требуется решить задачу 3.1 для матрицы L и набора Q.

Набор Q покрывает строки с номерами $1, \ldots, M, M+1$, при этом строки с номерами $1, \ldots, M$ являются конкурентными. Поэтому если верхний набор Q' содержит Q, то в $H(Q')$ должны входить столбцы с номерами $3, \ldots, 2+M$, чтобы строки с номерами $1, \ldots, M$ не были конкурентными для Q'. Из этого следует, что набор Q' содержит набор $\{(i_1, 3), \ldots, (i_M, 2+M)\}$, где для любого $l \in \{1, \ldots, M\}$ строка с номером i_l соответствует либо ребру $\{C_l, x_{s_l}\}$, либо ребру $\{C_l, \bar{x}_{s_l}\}$.

При этом Q' покрывает соответствующие ребрам $\{x_1, \bar{x}_1\}, \ldots, \{x_N, \bar{x}_N\}$ строки тогда и только тогда, когда среди номеров строк $i_1, \ldots, i_M$ нет двух таких i_u, i_v, что строка с номером i_u соответствует ребру $\{C_u, x_{s_u}\}$, а строка с номером i_v — ребру $\{C_v, \bar{x}_{s_v}\}$, и $s_u = s_v$.

Таким образом, существование верхнего набора $Q' \supset Q$ равносильно существованию набора значений переменных $x_1, \ldots, x_N$, для которого истинны все элементарные дизъюнкции $C_1, \ldots, C_M$, а следовательно, выполнима функция $f(x_1, \ldots, x_N)$.

В результате задача проверки выполнимости функции, заданной КНФ, полиномиально сведена к задаче 3.1. Как известно, задача проверки выполнимости КНФ NP-полна. ■

3.2.5. Новые алгоритмы дуализации АО1К, АО1М, АО2К, АО2М

Введем дополнительные понятия и обозначения. Пусть Q — совместимый набор элементов из $E(L)$ и элемент $(i, j) \in Q$. Строку t матрицы L будем называть *опорной* для пары (Q, j), если $a_{tj} = 1$ и $a_{tl} = 0$, $\forall l \in H(Q), \quad l \neq j$. Из совместимости Q следует, что для любого элемента (i, j) из Q строка i является опорной для (Q, j).

Строку t матрицы L будем называть *конкурентной* для совместимого набора Q, если найдется элемент $(i, j) \in Q$ такой, что строка t является опорной для (Q, j), и $t < i$. Ясно, что покрывающий набор Q является верхним тогда и только тогда, когда ни одна строка матрицы L не является конкурентной для Q.

Пусть $L(R, C)$ — подматрица матрицы L. Число $v_j(R) = \sum_{i \in R} a_{ij}$, $j \in C$, будет называть *весом* столбца j в подматрице $L(R, C)$. Число $w_i(C) = \sum_{j \in C} a_{ij}$, $i \in R$, будет называть *весом* строки i в подматрице $L(R, C)$. При $w_i(C) = 0$ ($v_j(R) = 0$) строку $i \in R$ (столбец $j \in C$) будем называть *нулевой* (*нулевым*) в подматрице $L(R, C)$. Очевидно, если строка i охватывает строку t в подматрице $L(R, C)$, то $w_i(C) \geqslant w_t(C)$.

Алгоритмы АО1К, АО1М, АО2К и АО2М. На шаге 1 на итерации 1 выполняется следующее.

1. Строится подматрица $L(R[\varnothing], C[\varnothing])$, совпадающая с матрицей L.
2. *[Только для АО1К и АО2К.]* Берется строка $t = 1$.
3. *[Только для АО1М и АО2М.]* Ищется строка t подматрицы $L(R[\varnothing], C[\varnothing])$ с наименьшим весом $w_t(C[\varnothing])$.
4. Строятся набор D столбцов, покрывающих строку t, и набор $B[\varnothing] = E(L(R[\varnothing], D))$.

5. Корень становится текущей вершиной, и происходит переход к следующей итерации.

Пусть на шаге $s, s \geqslant 1$, на итерации $t, t \geqslant 1$, текущей стала вершина Q. Тогда на итерации $t+1$ выполняется следующее.

1–3. Выполняются пункты 1–3 итерации $t+1$ алгоритма АО1.

4. *[Только для АО2К и АО2М.]* Из подматрицы $L(R[Q'], C[Q'])$ последовательно удаляются охватывающие снизу строки.

5. *[Только для АО1К и АО2К.]* Если набор Q' имеет хотя бы одну конкурентную строку, берется конкурентная строка t с наименьшим номером. В противном случае берется строка t подматрицы $L(R[Q'], C[Q'])$ с наименьшим номером.

6. *[Только для АО1М и АО2М.]* Среди строк подматрицы $L(R[Q'], C[Q'])$ и конкурентных для набора Q' строк ищется строка t с наименьшим весом $w_t(C[Q'])$.

7. Строятся набор D столбцов подматрицы $L(R[Q'], C[Q'])$, покрывающих строку t, и набор элементов $B[Q'] = E(L(R[Q'], D))$.

8. *[Только для АО2К и АО2М.]* Из набора $B[Q']$ удаляются элементы, запрещённые в подматрице $L(R[Q'], C[Q'])$.

9. Текущей вершиной становится Q', и происходит переход к следующей итерации.

На шаге $s, s > 1$, на итерации 1 выполняет те же действия, которые описаны в общей схеме работы алгоритма первого типа. ■

Всякий раз при формировании набора $B[Q]$ используются элементы набора $E(L(R[Q], D))$, где подматрица $L(R[Q], D)$ содержит только те столбцы, которые покрывают некоторую строку t, выбираемую различными алгоритмами по-разному. Поэтому число внутренних вершин дерева решений у алгоритмов АО1К и АО1М (АО2К и АО2М), как правило, меньше числа внутренних вершин дерева решений алгоритма АО1 (АО2).

Стратегию выбора строки в алгоритмах АО1К и АО2К можно объяснить следующим образом. Чем «раньше» при построении ветви дерева решений предпринимаются попытки повторно покрыть конкурентную строку, тем раньше может обнаружиться, что из текущей вершины ни одна ветвь дерева решений не заканчивается

верхним набором, то есть алгоритмы АО1К и АО2К «борются с конкурентными строками». Эксперименты показали, что деревья решений алгоритмов АО1К и АО2К, как правило, имеет меньше внутренних и «лишних» висячих вершин, чем деревья решений алгоритмов АО1 и АО2 соответственно.

Стратегию выбора строки в алгоритмах АО1М и АО2М можно объяснить следующим образом. Чем меньше вес $w_t(C)$ строки t, тем «сложнее» покрыть строку t столбцами текущей подматрицы $L(R, C)$, то есть алгоритмы АО1М и АО2М действуют «от сложного к простому». Оправдана эта стратегия тем, что результат шага должен покрывать все, в том числе и «сложные» строки, а покрыть «простые» строки, вероятнее всего, получится и позднее, даже после того, как в текущей подматрице $L(R, C)$ останется меньше столбцов. Заметим, что «самой сложной» для покрытия является нулевая строка, при обнаружении которых, фактически, алгоритм обрывает построение текущей ветви дерева решений. Алгоритмы АО1М и АО2М можно назвать «жадными» алгоритмами, минимизирующими число внутренних вершин дерева решений. Эксперименты показали, что дерево решений алгоритма АО1М (АО2М), как правило, имеет меньше внутренних и «лишних» висячих вершин, чем деревья решений алгоритмов АО1 и АО1К (АО2 и АО2К).

Для реализации описанной схемы алгоритмов дуализации АО1К, АО1М, АО2К и АО2М удобно использовать механизм рекурсивных вызовов. Выделим две процедуры: `BuildSubtreeAO` и `CreateNodeAO`. Процедура `BuildSubtreeAO` является рекурсивной и выполняет построение поддерева дерева решений. Процедура `CreateNodeAO` выполняет построение новой вершины Q дерева решений и обновляет состояние алгоритма: набор строк, не покрытых набором Q, набор строк, конкурентных для Q, набор столбцов, совместимых с набором Q, набор запрещённых элементов.

Введем глобальные переменные: Q — текущий совместимый набор элементов из $E(L)$, $L(R, C)$ — текущая подматрица, K — текущий набор конкурентных строк, Z — текущий набор запрещённых элементов.

Дуализация матрицы L алгоритмом АО1К (АО1М, АО2К, АО2М) начинается с инициализации глобальных переменных и вызова основной рекурсивной процедуры `BuildSubtreeAO` (см. алгоритм 2).

Алгоритм 2 Алгоритмы дуализации AO1K, AO1M, AO2K и AO2M

Вход: L — булева матрица размера $m \times n$;

Выход: $\mathcal{P}(L)$ — набор неприводимых покрытий матрицы L;

1: **глобальные переменные:**
   Q — текущий набор элементов из $E(L)$; $L(R,C)$ — текущая подматрица;
   K — текущий набор конкурентных строк; Z — текущий набор запрещённых элементов;
2: $Q := \varnothing$; $K := \varnothing$; $Z := \varnothing$; $R := \{1,\ldots,m\}$; $C := \{1,\ldots,n\}$;
3: **вызвать** `BuildSubtreeAO`; *// построение дерева решений с корнем* $\varnothing$

1: **ПРОЦЕДУРА** `BuildSubtreeAO`
2: *выбрать строку* t *из* $K \cup R$; *// [AO1K, AO2K]* t *— первая по порядку строка из*
   // K *при* $K \neq \varnothing$ *или из* R *при* $K = \varnothing$;
   // [AO1M, AO2M] t *— строка с минимальным весом* $w_t(C)$;
3: **для всех** $j \in C : a_{tj} = 1$
4: **для всех** $i \in R : a_{ij} = 1$
5: **если** $(i,j) \notin Z$ **то**
6: **вызвать** `CreateNodeAO`(i,j);
7: **если** $(R \neq \varnothing)$ **то вызвать** `BuildSubtreeAO`; *// рекурсивный вызов*
8: **если** $(R = \varnothing$ *и* $K = \varnothing)$ **то выдать** *неприводимое покрытие* $H(Q)$;
9: *убрать изменения, внесенные в глобальные переменные на шаге 6;*
10: *удалить* j *из* C;

На шаге 2 процедуры `BuildSubtreeAO` осуществляется выбор строки t, которая определяет ветвление в текущей вершине Q дерева решений. Каждая дочерняя вершина $Q \cup \{(i,j)\}$ удовлетворяет трем условиям: 1) элемент $(i,j) \in E(L(R,C))$; 2) столбец j покрывает стоку t; 3) элемент (i,j) не запрещен в $L(R,C)$. Как уже было сказано выше, алгоритмы AO1K, AO1M, AO2K и AO2M по-разному выбирают строку t, поэтому имеют различные деревья решений.

Построение дочерней вершины происходит при вызове процедуры `CreateNodeAO` на шаге 6, которая модифицирует текущее состояние алгоритма. Перед построением следующей дочерней вершины на шаге 9 восстанавливается состояние алгоритма до вызова процедуры `CreateNodeAO`.

Процедура `CreateNodeAO` имеет два параметра i и j, задающие элемент, добавляемый в текущий набор Q. На шаге 2 из K удаляются покрытые столбцом j строки, поскольку эти строки являются опорными для Q, но не являются опорными для $Q \cup \{i,j\}$. На шаге 3 в K добавляются покрытые столбцом j строки из R, лежащие выше i. Эти строки являются конкурентными для $Q \cup \{i,j\}$, поскольку

1: **ПРОЦЕДУРА** `CreateNodeAO`(i, j)

2: удалить из K строки, покрытые j;
3: добавить в K строки набора R, покрытые j и лежащие выше i;
4: удалить из R строки, покрытые j;
5: удалить из C столбцы, покрывающие i;
6: добавить (i, j) в Q;
// [AO2K и AO2M] удалить охватывающие снизу строки и найти запрещённые элементы:
7: **для всех** $s \in R$
8: **для всех** $t \in R : t > s$
9: **если** t *охватывает* s *в подматрице* $L(R, C)$ **то**
10: *удалить* t *из* R;
11: **иначе если** s *охватывает* t *в подматрице* $L(R, C)$ **то**
12: *добавить в* Z *запрещённые элементы вида* (s, l);

они не покрыты набором столбцов $H(Q)$, но покрыты столбцом j. На шаге 4 из R удаляются строки, покрытые столбцом j. На шаге 5 из C удаляются столбцы, покрывающие строку i, поскольку такие столбцы не совместимы с $Q \cup \{i, j\}$. Шаги 7–12 выполняются только в алгоритмах AO2K и AO2M. При этом из подматрицы $L(R, C)$ удаляются охватывающие снизу строки, и в набор B добавляются запрещённые в $L(R, C)$ элементы.

Оценим сложность шага алгоритмов AO1K и AO1M. Сложность процедуры `CreateNodeAO` равна $\mathcal{O}(mn)$. Сложность выполнения шага 2 в процедуре `BuildSubtreeAO` алгоритмом AO1M равна $\mathcal{O}(mn)$ Число рекурсивных вызовов `BuildSubtreeAO` для выполнения шага алгоритма (глубина дерева решений) не превосходит q, где $q = \min\{m, n\}$. Таким образом, сложность шага алгоритма AO1K и AO1M равна $\mathcal{O}(mnq)$. Напомним, что сложность алгоритма AO1 также равна $\mathcal{O}(mnq)$. Для работы алгоритма дополнительно требуется $\mathcal{O}(m + n)$ памяти для текущих наборов строк и столбцов подматрицы и текущего набора конкурентных строк.

Оценим сложность шага алгоритмов AO2K и AO2M. Сложность процедуры `CreateNodeAO` равна $\mathcal{O}(m^2 n)$ в связи с удалением охватывающих строки и обновления множества запрещённых элементов (см. цикл 7–12). Число рекурсивных вызовов `BuildSubtreeAO` для выполнения шага алгоритма (глубина дерева решений) не превосходит q. Таким образом, сложность шага алгоритма AO2K и AO2M

равна $\mathcal{O}(m^2nq)$. Для работы алгоритма дополнительно требуется $\mathcal{O}(mn)$ памяти для хранения текущего набора запрещённых элементов.

3.3. Асимптотически оптимальные алгоритмы дуализации второго типа

Даётся схема асимптотически оптимального алгоритма дуализации второго типа. В рамках схемы описываются ранее построенные алгоритмы ОПТ, RS и MMCS, а также новые алгоритмы RUNC, RUNC-M и PUNC. Для этого вводятся дополнительные понятия и обозначения.

3.3.1. Общая схема алгоритмов дуализации второго типа

Асимптотически оптимальные алгоритмы дуализации второго типа перечисляют с полиномиальной задержкой без повторений максимальные совместимые наборы столбцов матрицы L. Опишем общую схему работы алгоритма второго типа. Введем используемые далее понятия и обозначения.

Пусть H — набор столбцов матрицы L. Строка i матрицы L называется *опорной* для пары (H, j), $j \in H$, если $a_{ij} = 1$ и $a_{il} = 0$, $l \neq j$, $l \in H$. Очевидно, набор H является совместимым тогда и только тогда, когда для каждого (H, j), $j \in H$, существует хотя бы одна опорная строка. Множество всех опорных строк для (H, j) обозначим через $S(H, j)$.

Столбец j матрицы L называется *запрещённым* для набора столбцов H, если существует столбец $l \in H$ такой, что столбец j покрывает все опорные для (H, l) строки. В противном случае будем говорить, что столбец j *совместим* с набором H. Очевидно, что $H \cup \{j\}$ является совместимым набором тогда и только тогда, когда столбец j совместим с набором H.

Работу асимптотически оптимального алгоритма дуализации второго типа можно представить в виде одностороннего обхода ветвей дерева решений, вершины которого, за исключением корня, — совместимые наборы столбцов. Корень дерева — пустой набор столбцов. Висячие вершины либо являются неприводимыми покрытиями, либо соответствуют лишним шагам алгоритма.

Каждый шаг алгоритма является итеративной процедурой, в результате которой строится одна ветвь дерева, начинающаяся либо в корне, либо в некоторой постро-

енной ранее внутренней вершине. При переходе от вершины к вершине меняется состояние алгоритма. Для обозначения того, что некоторый объект X, описывающий состояние алгоритма, связан с вершиной H будем писать $X[H]$.

Общая схема асимптотически оптимального алгоритма дуализации второго типа. На шаге 1 на итерации 1 по некоторому правилу формируется набор столбцов $C[\varnothing]$ матрицы L, корень становится текущей вершиной, и происходит переход к следующей итерации.

Пусть на шаге $s, s \geqslant 1$, на итерации $t, t \geqslant 1$, текущей стала вершина H. Тогда на итерации $t+1$ выполняется следующее.

1. Если $C[H] = \varnothing$, то происходит переход к следующему шагу (в случае, когда H является висячей вершиной, шаг алгоритма считается лишним). В противном случае берется первый по порядку столбец $j \in C[H]$ и удаляется из $C[H]$.

2. Если j является запрещённым для H, то текущая вершина не меняется, и происходит переход к следующей итерации. В противном случае строится вершина $H' = H \cup \{j\}$.

3. Если столбец j покрывает строки, не покрытые набором H, то результатом шага становится неприводимое покрытие H', и происходит переход к следующему шагу. В противном случае по некоторому правилу строится набор столбцов $C[H']$.

4. Текущей вершиной становится H', и происходит переход к следующей итерации.

Пусть результатом шага $s, s \geqslant 1$, является набор H. Тогда на шаге $s+1$ на итерации 1 среди вершин ветки дерева, соединяющей корень с вершиной H, ищется ближайшая к H вершина H' такая, что $C[H'] \neq \varnothing$. Если вершина H' найдена, то она становится текущей вершиной, и происходит переход к следующей итерации. В противном случае алгоритм завершает работу. ■

Алгоритмы второго типа различаются правилом построения набора $C[\varnothing]$ на шаге 1 на итерации 1 и правилом построения набора $C[H]$ при создании новой вершины H. Опишем эти различия на примере алгоритмов ОПТ, MMCS и RS.

3.3.2. Алгоритм ОПТ

На шаге 1 на итерации 1 строится подматрица $L(R[\varnothing], C[\varnothing])$ путем последовательного удаления из матрицы L охватывающих строк и нулевых столбцов. Далее корень становится текущей вершиной, и происходит переход к следующей итерации.

Пусть на шаге $s, s \geqslant 1$, на итерации $t, t \geqslant 1$, текущей стала вершина H. Тогда на итерации $t+1$ выполняется следующее.

1. Если $C[H] = \varnothing$, то происходит переход к следующему шагу. В противном случае берется первый по порядку столбец $j \in C[H]$ и удаляется из $C[H]$.

2. Строится вершина $H' = H \cup \{j\}$.

3. Если столбец j покрывает строки, не покрытые набором H, то результатом шага становится неприводимое покрытие H', и происходит переход к следующему шагу. В противном случае строится подматрица $L(R[H'], C[H'])$ путем удаления из подматрицы $L(R[H], C[H])$ строк, покрытые столбцом j, и столбцов, запрещённые для набора H'. Затем из $L(R[H'], C[H'])$ последовательно удаляются охватывающие строки и нулевые столбцы.

4. Текущей вершиной становится H', и происходит переход к следующей итерации.

На шаге $s, s > 1$, на итерации 1 алгоритм ОПТ выполняет те же действия, которые описаны в общей схеме работы алгоритма второго типа. ■

В алгоритме ОПТ при построении новой вершины H в подматрицу $L(R[H], C[H])$ включаются только совместимые с H и ненулевые столбцы. Поэтому каждый столбец подматрицы $L(R[H], C[H])$ порождает дочернюю для H вершину, то есть нет необходимости проверять, является ли столбец, выбранный из текущей подматрицы, совместимым с текущим набором.

3.3.3. Алгоритм MMCS

На шаге 1 на итерации 1 выбирается строка i матрицы L, строится набор столбцов $C[\varnothing]$, покрывающих строку i. Далее из столбцов, не входящих в $C[\varnothing]$, строится подматрица $L(R[\varnothing], D[\varnothing])$, корень становится текущей вершиной, и происходит переход к следующей итерации.

Пусть на шаге $s, s \geqslant 1$, на итерации $t, t \geqslant 1$, текущей стала вершина H. Тогда на итерации $t+1$ выполняется следующее.

1. Если $C[H] = \varnothing$, то происходит переход к следующему шагу. В противном случае берется первый по порядку столбец $j \in C[H]$ и удаляется из $C[H]$.

2. Если j является запрещённым для H, то текущая вершина не меняется, и происходит переход к следующей итерации. В противном случае строится вершина $H' = H \cup \{j\}$, и столбец j добавляется в $D[H]$.

3. Если столбец j покрывает строки, не покрытые набором H, то результатом шага становится неприводимое покрытие H', и происходит переход к следующему шагу. В противном случае в подматрице $L(R[H], D[H])$ выбирается строка i, не покрытая столбцом j, формируется набор $C[H']$ покрывающих строку i столбцов подматрицы $L(R[H], D[H])$, и строится подматрица $L(R[H'], D[H'])$ путем удаления из подматрицы $L(R[H], D[H])$ строк, покрытых столбцом j, и столбцов набора $C[H']$.

4. Текущей вершиной становится H', и происходит переход к следующей итерации.

На шаге $s, s > 1$, на итерации 1 алгоритм MMCS выполняет те же действия, которые описаны в общей схеме работы алгоритма второго типа. ■

В алгоритме MMCS при построении новой вершины H выбирается некоторая строка i в подматрице $L(R[H], D[H])$. Для вершины H строятся только такие дочерние вершины $H \cup \{j\}$, для которых столбец j подматрицы $L(R[H], D[H])$ покрывает выбранную строку i. Такой способ ветвления позволяет, как правило, строить дерево решений с меньшим числом внутренних вершин, чем у алгоритма ОПТ (в алгоритме ОПТ все столбцы подматрицы $L(R[H], C[H])$ используются для построения дочерних вершин).

3.3.4. Алгоритм RS

Для описания алгоритма RS потребуются дополнительные понятия и обозначения.

Пусть H — набор столбцов матрицы L и пусть $i \in \{1, \ldots, m\}$. Обозначим набор строк, опорных для (H, j), $j \in H$, лежащих не ниже строки i, через $S^i(H, j)$. Набор столбцов H называется *i-совместимым*, если H покрывает строки $1, 2, \ldots, i$,

и $S^i(H,j) \neq \varnothing$, $\forall j \in H$. Пустой набор столбцов будем называть 0-совместимым. Заметим, что m-совместимый набор столбцов — неприводимое покрытие матрицы L.

Пусть $i \geqslant 1$, набор столбцов H не покрывает строку i и является $(i-1)$-совместимым. Столбец j называется *i-запрещённым* для набора H, если либо столбец j не покрывает строку i, либо $i > 1$ и в H найдется столбец l такой, что столбец j покрывает все строки из $S^{i-1}(H,l)$.

Заметим, что в случае, когда столбец j не является i-запрещённым для H, набор $H \cup \{j\}$ обладает свойством i-совместимости, а следовательно совместим. В противном случае $H \cup \{j\}$ не является i-совместимым и в зависимости от строк $i+1, i+2, \ldots, m$ может быть несовместимым. При этом для i-запрещённого столбца j может существовать строка $t > i$ такая, что j не является t-запрещённым.

Алгоритм RS. На шаге 1 на итерации 1 строится подматрица $L(R[\varnothing], C[\varnothing])$, состоящая из столбцов матрицы L, покрывающих первую строку. Корень становится текущей вершиной, и происходит переход к следующей итерации.

Пусть на шаге $s, s \geqslant 1$, на итерации $t, t \geqslant 1$, текущей стала вершина H. Тогда на итерации $t+1$ выполняется следующее.

1. Если $C[H] = \varnothing$, то происходит переход к следующему шагу. В противном случае берется первый по порядку столбец $j \in C[H]$ и удаляется из $C[H]$.

2. Строится вершина $H' = H \cup \{j\}$.

3. Если столбец j покрывает строки, не покрытые набором H, то результатом шага становится неприводимое покрытие H', и происходит переход к следующему шагу. В противном случае в подматрице $L(R[H], C[H])$ выбирается первая по порядку строка i, не покрытая столбцом j, формируется набор столбцов $C[H']$, не являющихся i-запрещёнными для H', и строится подматрица $L(R[H'], C[H'])$, содержащая строки подматрицы $L(R[H], C[H])$, не покрытые столбцом j.

4. Текущей вершиной становится H', и происходит переход к следующей итерации.

На шаге $s, s > 1$, на итерации 1 алгоритм RS выполняет те же действия, которые описаны в общей схеме работы алгоритма второго типа. ■

В алгоритме RS при построении новой вершины H выбирается не покрытая набором H строка i с наименьшим номером. Нарушение правила выбора строки приводит к потере решений. Для вершины H строятся только такие дочерние вершины $H \cup \{j\}$, для которых столбец j матрицы L покрывает выбранную строку i и не нарушает свойство i-совместимости. Такой способ ветвления позволяет строить дерево решений, число внутренних вершин которого, как правило, меньше чем у алгоритма ОПТ, но большим, чем у алгоритма MMCS.

3.3.5. Новые алгоритмы дуализации RUNC и RUNC-M

В данном разделе строятся новые асимптотически оптимальные алгоритмы дуализации второго типа RUNC, RUNC-M. Алгоритмы описываются в рамках общей схема. Приводится способ их реализации с использованием рекурсивной процедуры. Анализируется сложность шага алгоритмов.

В построенных алгоритмах RUNC (аббревиатура от Remove Unallowable Columns) и RUNC-M (RUNC Minimum weight row) комбинируются идеи алгоритмов ОПТ и MMCS. Различия алгоритмов RUNC и RUNC-M незначительны, поэтому ниже дается их общее описание, и делаются замечания относительно особенностей работы каждого алгоритма.

Алгоритмы RUNC и RUNC-M. На шаге 1 на итерации 1, аналогично алгоритму MMCS, выбирается строка i матрицы L (алгоритм RUNC использует $i = 1$, алгоритм RUNC-M ищет строку i с минимальным весом в матрице L), строится набор столбцов $C[\varnothing]$, покрывающих строку i, и строится подматрица $L(R[\varnothing], D[\varnothing])$ путем последовательного удаления из матрицы L охватывающих строк и нулевых столбцов. Далее корень становится текущей вершиной, и происходит переход к следующей итерации.

Пусть на шаге $s, s \geqslant 1$, на итерации $t, t \geqslant 1$, текущей стала вершина H. Тогда на итерации $t + 1$ выполняется следующее.

1. Если $C[H] = \varnothing$, то происходит переход к следующему шагу. В противном случае берется первый по порядку столбец $j \in C[H]$, столбец j удаляется из $C[H]$ и из $D[H]$.

2. Строится вершина $H' = H \cup \{j\}$.

3. Если столбец j покрывает строки, не покрытые набором H, то результатом шага становится неприводимое покрытие H', и происходит переход к следующему шагу. В противном случае в подматрице $L(R[H], D[H])$ выбирается строка i, не покрытая столбцом j (алгоритм RUNC использует строку с наименьшим номером, алгоритм RUNC-M ищет строку i с наименьшим весом $w_i(D[H])$), формируется набор $C[H']$ покрывающих строку i столбцов подматрицы $L(R[H], D[H])$, и строится подматрица $L(R[H'], D[H'])$ путем удаления из подматрицы $L(R[H], D[H])$ покрытых столбцом j строк и, аналогично алгоритму ОПТ, запрещённых для H' столбцов.

4. Текущей вершиной становится H', и происходит переход к следующей итерации.

На шаге $s, s > 1$, на итерации 1 алгоритм RUNC (RUNC-M) выполняет те же действия, которые описаны в общей схеме работы алгоритма второго типа. ■

Фактически, алгоритмы RUNC и RUNC-M различаются способом выбора строки i, определяющей состав дочерних вершин в дереве решений для новой построенной вершины H. Экспериментально установлено, что в большинстве случаев RUNC-M является эффективнее RUNC, поскольку его дерево решений имеет существенно меньше внутренних вершин, что компенсирует вычислительные затраты поиска строки с минимальным весом. Алгоритм RUNC-M можно назвать «жадным» алгоритмом, минимизирующим число внутренних вершин дерева решений.

Алгоритм RUNC работает быстрее RUNC-M в тех случаях, когда матрица является разреженной и в каждой строке примерно одинаковое число единиц. В таких входных матрицах выбор строки с минимальным весом в алгоритме RUNC-M не влияет или мало влияет на число внутренних вершин дерева решений, а затраты на поиск строки с минимальным весом могут оказаться существенными.

Для реализации описанной схемы алгоритмов дуализации RUNC и RUNC-M удобно использовать механизм рекурсивных вызовов. Выделим две процедуры `BuildSubtreeRUNC` и `CreateNodeRUNC`. Процедура `BuildSubtreeRUNC` является рекурсивной и выполняет построение поддерева дерева решений. Процедура `CreateNodeRUNC` выполняет построение новой вершины H дерева решений и обновляет состояние алгоритма: набор строк, не покрытых набором H, набор столбцов, совместимых с H, и наборы строк, опорных для $(H, j), j \in H$.

Алгоритм 3 Алгоритмы дуализации RUNC и RUNC-M

Вход: L — булева матрица размера $m \times n$;

Выход: $\mathcal{P}(L)$ — набор неприводимых покрытий матрицы L;

1: **глобальные переменные:**
   H — текущий набор столбцов; $L(R, D)$ — текущая подматрица;
   S_l, $l \in \{1, \ldots, n\}$, — текущий набор опорных для (H, l) строк;
2: $H := \varnothing$; $R := \{1, \ldots, m\}$; $D := \{1, \ldots, n\}$;
3: **для всех** $l = 1, \ldots, n$
4: $\quad S_l := \varnothing$;
5: **вызвать** `BuildSubtreeRUNC`; *// построение дерева решений с корнем* $\varnothing$

1: **ПРОЦЕДУРА** `BuildSubtreeRUNC`

2: *выбрать строку* $i \in R$; *// RUNC:* i *— первая по порядку строка в* R*;*
   // RUNC-M: i *— строка с минимальным весом* $w_i(D)$*;*
3: **для всех** $(j \in D : a_{ij} = 1)$
4: $\quad$ *удалить* j *из* D*;*
5: $\quad$ **вызвать** `CreateNodeRUNC`(j)*;*
6: $\quad$ **если** $(R = \varnothing)$ **то**
7: $\quad\quad$ **выдать** *неприводимое покрытие* H*;*
8: $\quad$ **иначе**
9: $\quad\quad$ **вызвать** `BuildSubtreeRUNC`*;* *// рекурсивное построение ветки дерева*
10: $\quad$ *убрать изменения, внесенные в глобальные переменные на шаге 5;*

Введем глобальные переменные: H — текущий совместимый набор столбцов, $L(R, D)$ — текущая подматрица, S_j, $j \in \{1, \ldots, n\}$, — текущий набор опорных строк для (H, j).

Дуализация матрицы L алгоритмом RUNC (RUNC-M) начинается с инициализации глобальных переменных и вызова основной рекурсивной процедуры `BuildSubtreeRUNC` (см. алгоритм 3).

На шаге 2 процедуры `BuildSubtreeRUNC` осуществляется выбор строки i, которая определяет ветвление в текущей вершине H дерева решений. Каждая дочерняя вершина $H \cup \{j\}$ удовлетворяет условию: столбец j покрывает стоку i. Как уже было сказано выше, алгоритмы RUNC и RUNC-M по-разному выбирают строку i, поэтому имеют различные деревья решений.

Построение дочерней вершины происходит при вызове процедуры `CreateNodeRUNC` на шаге 5, которая модифицирует текущее состояние алгорит-

```
1: ПРОЦЕДУРА CreateNodeRUNC(j)
2: S_j := {i ∈ R : a_ij = 1};                          // добавить новые опорные строки
3: удалить из R строки, покрытые j;
4: для всех (l ∈ H : S_l ≠ ∅)
5:     удалить из S_l строки, покрытые столбцом j;      // обновить опорные строки
6:     если (∃i ∈ S_l : a_ip = 0, ∀p ∈ D) то
7:         S_l := ∅;
8:     иначе
9:         для всех (p ∈ D)
10:            если (a_ip = 1, ∀i ∈ S_l) то
11:                удалить p из D;                      // удалить запрещённые столбцы
12: добавить j в H;
```

ма. Перед построением следующей дочерней вершины на шаге 10 восстанавливается состояние алгоритма до вызова процедуры `CreateNodeRUNC`.

Процедура `CreateNodeRUNC` имеет единственный параметр j — столбец, добавляемый в текущий набор столбцов H. На шаге 2 процедуры `CreateNodeRUNC` набор строк из R, покрытых столбцом j, становится текущим набором опорных строк S_j (эти строки являются опорными для $(H \cup \{j\}, j)$, поскольку их не покрывает набор H). Далее из R удаляются строки, покрытые столбцом j. Для каждого $l \in H$ такого, что текущий набор опорных для (H, l) строк не пуст, выполняются шаги 5–11. На шаге 5 из S_l удаляются строки, покрытые столбцом j, так как эти строки не являются опорными для $(H \cup \{j\}, l)$. В случае, когда текущий набор S_l, $l \in H \cup \{j\}$, содержит хотя бы одну строку, не покрытую набором столбцов D, набор S_l не влияет на состав запрещённых столбцов в D, поэтому на шаге 7 набор S_l заменяется на $\varnothing$. На шагах 9–11 из D удаляются запрещённые для $H \cup \{j\}$ столбцы.

Оценим сложность шага алгоритма RUNC (RUNC-M). Без учета рекурсивных вызовов сложность процедур `BuildSubtreeRUNC` и `CreateNodeRUNC` равна $\mathcal{O}(mn)$. Глубина дерева решений не превосходит q, где $q = \min\{m, n\}$. Таким образом, сложность шага алгоритма RUNC (RUNC-M) равна $\mathcal{O}(mnq)$. Для работы алгоритма дополнительно требуется $\mathcal{O}(m + n)$ памяти.

3.3.6. Новый алгоритм дуализации PUNC

В данном разделе строится новый асимптотически оптимальный алгоритм дуализации второго типа PUNC. Алгоритм описывается в рамках общей схема. Приводится способ его реализации с использованием рекурсивной процедуры. Анализируется сложность шага алгоритма.

В основе построенного алгоритма PUNC (аббревиатура от Pending Unallowable Columns) лежит идея алгоритма RS.

Алгоритм PUNC. На шаге 1 на итерации 1 с корнем связывается строка $i[\varnothing] = 1$, и строится набор столбцов $C[\varnothing]$ матрицы L, покрывающих строку 1. Корень становится текущей вершиной, и происходит переход к следующей итерации.

Пусть на шаге $s, s \geqslant 1$, на итерации $t, t \geqslant 1$, текущей стала вершина H. Тогда на итерации $t+1$ выполняется следующее.

1. Если $C[H] = \varnothing$, то происходит переход к следующему шагу. В противном случае берется первый по порядку столбец $j \in C[H]$.

2. Строится вершина $H' = H \cup \{j\}$.

3. Ищется первая по порядку строка i', $i[H] < i' \leqslant m$, не покрытая столбцом j. Если найти такую строку не удается, то результатом шага становится неприводимое покрытие H', и происходит переход к следующему шагу. В противном случае формируется набор столбцов $C[H']$, не являющихся i'-запрещёнными для H', с вершиной H' связывается строка $i[H'] = i'$.

4. Текущей вершиной становится H', и происходит переход к следующей итерации.

На шаге $s, s > 1$, на итерации 1 алгоритм PUNC выполняет те же действия, которые описаны в общей схеме работы алгоритма второго типа. ■

Для реализации описанной схемы алгоритма дуализации PUNC удобно использовать механизм рекурсивных вызовов. Выделим две процедуры: `BuildSubtreePUNC` и `CreateNodePUNC`. Процедура `BuildSubtreePUNC` является рекурсивной и выполняет построение поддерева дерева решений. Процедура `CreateNodePUNC` выполняет построение новой вершины H дерева решений и

Алгоритм 4 Алгоритм дуализации PUNC

Вход: L — булева матрица размера $m \times n$;

Выход: $\mathcal{P}(L)$ — набор неприводимых покрытий матрицы L;

1: **глобальные переменные:**
   H — текущий совместимый набор столбцов;
   i — текущая строка, не покрытая H;
   S_l, $l \in \{1, \ldots, n\}$, — текущий набор опорных для (H, l) строк;
2: $H := \varnothing$;
3: $i := 1$;
4: **для всех** $l = 1, \ldots, n$
5: $S_l := \varnothing$;
6: **вызвать** `BuildSubtreePUNC`; *// построение дерева решений с корнем* $\varnothing$

1: **ПРОЦЕДУРА** `BuildSubtreePUNC`

2: **для всех** $(j : a_{ij} = 1)$
3: **если** $(\forall l \in H,\ \exists t \in S_l : a_{tj} = 0)$ **то**
4: `CreateNodePUNC`(j);
5: **если** $(i > m)$ **то**
выдать *неприводимое покрытие* H;
6: **иначе**
7: **вызвать** `BuildSubtreePUNC`; *//рекурсивный вызов*
8: *убрать изменения, внесенные в глобальные переменные на шаге 4;*

обновляет текущее состояние алгоритма: номер строки, не покрытой набором H и набор строк, опорных для $(H, j), j \in H$.

Введем глобальные переменные: H — текущий совместимый набор столбцов, i — текущая строка, не покрытая набором H, S_j, $j \in \{1, \ldots, n\}$, — текущий набор опорных строк для (H, j).

Дуализация матрицы L алгоритмом PUNC начинается с инициализации глобальных переменных и вызова основной рекурсивной процедуры `BuildSubtreePUNC` (см. алгоритм 4).

В вроцедуре `BuildSubtreePUNC` среди столбцов, покрывающих текущую строку i перебираются i-совместимые с текущим набором H столбцы. Проверка i-совместимости осуществляется на шаге 3 с использованием текущих наборов опорных строк S_l, $l \in H$. Для каждого i-совместимого столбца j на шаге 4 вызывается процедура `CreateNodePUNC` для построения вершины $H \cup \{j\}$, и на шаге 7 ре-

1: **ПРОЦЕДУРА** `CreateNodePUNC(`j`)`

2: **для всех** $l \in H$

3: удалить из S_l строки, покрытые столбцом j;

4: добавить j в H;

5: **пока (**$i \leqslant m$ и набор H покрывает строку i**)**

6: **если (**$\exists l \in H$: строка i является опорной для (H, l)**) то**

7: добавить i в S_l;

8: $i := i + 1$;

курсивно вызывается процедура `BuildSubtreePUNC` для построения поддерева с корнем $H \cup \{j\}$.

При вызове `CreateNodePUNC(`j`)` текущей вершиной становится $H' = H \cup \{j\}$, обновляются текущая строка i, не покрытая H', и наборы строк, опорных для (H', l), $l \in H'$. При этом после вызова процедуры `CreateNodePUNC` имеют место равенства $S_l = S^i(H, l)$, $l \in H$.

Сравнивая описание алгоритма RS из [77] и подробное описание алгоритма PUNC, можно заметить следующее. Алгоритм RS после построения очередного набора H формирует (модифицирует) набор строк, не покрытых набором H, и наборы строк $S(H, l), l \in H$. Алгоритм PUNC после построения очередного набора H модифицирует минимальный номер строки i, не покрытой набором H, и наборы строк $S^i(H, l), l \in H$. Наборов $S^i(H, l), l \in H$, достаточно для проверки i-совместимости столбцов. При этом временные затраты алгоритма RS на построение и модификацию наборов $S(H, l), l \in H$, как правило, выше временных затрат алгоритма PUNC на построение и модификацию наборов $S^i(H, l), l \in H$.

Сложность выполнения одного шага алгоритма PUNC ограничена $\mathcal{O}(mnq)$, где $q = \min\{m, n\}$. Однако следует отметить, что шаг алгоритма PUNC в среднем выполняется быстрее шага алгоритмом RUNC и RUNC-M, поскольку при выполнении шага алгоритма PUNC для каждой строки i матрицы L проверка того, что строка i покрыта текущим набором столбцов H, осуществляется не более одного раза.

Таблица 3.1: Случайные матрицы. Случай $m = 10, 50 \leqslant n \leqslant 300$

ОПТ	**0.007**	0.06	0.31	1.2	3.2	8.
MMCS	0.008	0.1	0.6	2.3	6.4	15.4
RUNC	0.01	0.06	0.3	1.5	4.3	10.8
RUNC-M	0.011	0.06	0.3	1.4	4.	11.
RS	0.008	0.09	0.5	2.2	6.	14.4
PUNC	**0.007**	0.06	0.3	1.2	3.2	7.8
AO1	0.009	0.1	0.5	1.7	4.3	10.2
AO1K	**0.007**	**0.04**	**0.23**	**0.9**	2.4	5.9
AO1M	0.011	0.05	0.24	1.	2.5	6.1
AO2	0.01	0.09	0.5	1.7	4.3	10.1
AO2K	0.009	**0.04**	0.24	1.	**2.3**	**5.7**
AO2M	**0.007**	0.05	0.24	1.	2.5	6.1
m	10	10	10	10	10	10
n	50	100	150	200	250	300
$\lvert\mathcal{P}(L)\rvert^*$	8361	146248	836107	3257326	8666765	20658771
$\lvert H\rvert^*$	3.8	4.1	4.3	4.5	4.6	4.7

3.4. Экспериментальное исследование асимптотически оптимальных алгоритмов дуализации

Для тестирования эффективности предложенных в работе алгоритмов была проведена серия экспериментов на случайных булевых матрицах, на модельных данных и прикладных задачах.

Алгоритмы ОПТ, RUNC, RUNC-M и PUNC были реализованы на языке C++. Отметим одну особенность реализации этих алгоритмов. Элементы каждой строки булевой матрицы с n столбцами представлялись последовательностью битов массива из $\lceil n/32 \rceil$ двойных машинных слов. Такое представление позволило некоторые операции с конечными наборами строк и столбцов заменить на битовых операций, применяемые к двойным машинным словам. Исходные коды программ алгоритмов MMCS и RS взяты из http://research.nii.ac.jp/~uno/dualization.html.

Формирование набора случайных матриц осуществлялось аналогично [39]. Каждая случайная матрица L размера $m \times n$ формировалась с помощью датчика случайных чисел так, что каждый элемент a_{ij} с одинаковой вероятностью принимал значение 0 или 1. Для каждой пары (m, n) обсчитывалось по 20 случайных матриц $L_1, \ldots, L_{20}$, и вычислялось среднее число покрытий и средняя длина покрытия. Результаты счёта на случайных матрицах приведены в таблицах 3.1–3.7. Алгоритмы сравнивались в следующих случаях:

Таблица 3.2: Случайные матрицы. Случай $m = 20$, $50 < n < 150$

ОПТ	0.03	0.17	0.7	2.2	5.2
MMCS	0.04	0.28	1.1	4.1	10.2
RUNC	0.029	0.16	**0.6**	2.2	5.7
RUNC-M	**0.021**	**0.14**	**0.6**	2.	5.2
RS	0.04	0.27	1.1	3.8	9.2
ρUNC	0.03	0.16	**0.6**	2.	5.
AO1	0.12	0.6	2.	5.9	12.9
AO1K	0.04	0.2	0.7	2.1	5.
AO1M	0.03	0.17	**0.6**	**1.9**	**4.7**
AO2	0.09	0.5	1.7	5.3	11.7
AO2K	0.04	0.2	0.7	2.1	4.9
AO2M	0.03	0.17	**0.6**	2.	4.8
m	20	20	20	20	20
n	50	75	100	125	150
$\lvert\mathcal{P}(L)\rvert^*$	44552	314154	1269495	4254740	10321112
$\lvert H\rvert^*$	4.6	4.8	4.9	5.	5.1

Таблица 3.3: Случайные матрицы. Случай $m = 30$, $50 \leqslant n \leqslant 110$

ОПТ	0.08	0.4	1.8	4.8
MMCS	0.12	0.7	3.	9.1
RUNC	0.07	0.4	1.7	4.7
RUNC-M	**0.06**	**0.3**	**1.4**	**4.3**
RS	0.12	0.6	2.9	8.2
ρUNC	0.08	0.4	1.6	4.7
AO1	0.7	2.5	9.6	22.9
AO1K	0.2	0.8	3.	7.3
AO1M	0.1	0.5	2.	5.5
AO2	0.4	1.7	7.1	17.8
AO2K	0.16	0.7	2.7	6.7
AO2M	0.1	0.5	2.	5.5
m	30	30	30	30
n	50	70	90	110
$\lvert\mathcal{P}(L)\rvert^*$	113307	608535	2772442	7917863
$\lvert H\rvert^*$	5.	5.1	5.3	5.4

1) $n > m$ (см. табл. 3.1, 3.2, 3.3);

2) $n < m$ (см. табл. 3.4, 3.5, 3.6);

3) $n = m$ (см. табл. 3.7).

Столбцы таблиц 3.1–3.7 (кроме первого) соответствуют задачам, сгруппированным по размерам матриц. Параметры задачи приведены в нижних строках таблиц. В строках «m» и «n» указаны размеры матриц. В строках «$\lvert\mathcal{P}(L)\rvert^*$» и «$\lvert H\rvert^*$» указаны соответственно среднее число неприводимых покрытий и средняя длина неприводимого покрытия для матриц одного размера. Остальные строки таблицы содержат среднее время работы в секундах каждого из тестируемых алгоритмов на задачах соответствующего размера.

Таблица 3.4: Случайные матрицы. Случай $50 \leqslant m \leqslant 300$, $n = 40$

ОПТ	0.09	0.5	1.2	2.2	3.4	5.
MMCS	0.12	0.5	1.3	2.4	3.7	5.2
RUNC	0.07	0.3	0.9	1.6	2.4	3.5
RUNC-M	**0.06**	**0.24**	**0.6**	**1.**	**1.6**	**2.2**
RS	0.13	0.6	1.4	2.5	3.8	5.4
ρUNC	0.09	0.4	1.2	2.3	3.8	5.8
AO1	2.4	67.1	> 600	> 600	> 600	> 600
AO1K	0.7	17.3	154.1	> 600	> 600	> 600
AO1M	0.18	1.5	6.	14.6	30.1	55.2
AO2	0.9	9.8	46.8	134.1	273.5	> 600
AO2K	0.3	3.9	19.	56.1	114.6	223.9
AO2M	0.18	1.3	5.	12.	22.7	39.1
m	50	100	150	200	250	300
n	40	40	40	40	40	40
$\lvert\mathcal{P}(L)\rvert^*$	94389	333177	712507	1155969	1636729	2183946
$\lvert H\rvert^*$	5.5	6.3	6.8	7.1	7.4	7.6

Таблица 3.5: Случайные матрицы. Случай $70 \leqslant m \leqslant 230$, $n = 50$

ОПТ	0.8	2.4	5.5	9.7	15.9
MMCS	1.	3.1	6.9	11.8	19.1
RUNC	0.6	1.8	4.	7.	11.3
RUNC-M	**0.5**	**1.4**	**3.**	**5.1**	**8.**
RS	1.	3.	6.6	11.2	17.9
ρUNC	0.7	2.4	5.8	10.6	18.1
AO1	33.1	329.3	> 600	> 600	> 600
AO1K	9.3	90.6	> 600	> 600	> 600
AO1M	1.8	8.5	26.7	58.1	117.8
AO2	9.8	57.7	195.9	> 600	> 600
AO2K	4.	23.5	81.7	206.3	> 600
AO2M	1.6	7.9	23.4	49.5	97.7
m	70	110	150	190	230
n	50	50	50	50	50
$\lvert\mathcal{P}(L)\rvert^*$	705711	1813930	3593400	5682139	8450430
$\lvert H\rvert^*$	6.	6.5	6.9	7.1	7.4

При небольшом количестве строк $m = 10, 20$ среди алгоритмов нет однозначного лидера. В тройку самых быстрых входят алгоритмы ОПТ, ρUNC и RUNC-M (см. табл. 3.1 и 3.2). В остальных случаях алгоритм RUNC-M опережает другие алгоритмы. Причем преимущество в скорости алгоритма RUNC-M тем существенней, чем больше размер входной матрицы (см. табл. 3.3–3.7). Отставание алгоритмов ОПТ и RS от алгоритма RUNC-M объясняется тем, что время выполнения каждого шага этих алгоритмов сильно зависит от числа строк m входной матрицы.

На скорость работы рассматриваемых алгоритмов существенно влияет число вершин дерева решений. По этому показателю самым эффективным является алгоритм RUNC-M. Интересно, что дерево решений алгоритма ОПТ, делающего, как правило, наименьшее число лишних шагов, почти всегда состоит из наибольшего числа

Таблица 3.6: Случайные матрицы. Случай $90 \leqslant m \leqslant 150$, $n = 60$

ОПТ	4.8	8.5	13.1	19.
MMCS	7.	12.4	18.5	26.7
RUNC	3.7	6.5	10.1	14.2
RUNC-M	**2.9**	**5.1**	**7.9**	**10.9**
RS	6.5	11.2	17.1	23.9
ρUNC	4.8	8.7	13.8	20.8
AO1	303.6	> 600	> 600	> 600
AO1K	82.5	254.	> 600	> 600
AO1M	13.4	28.9	54.2	88.3
AO2	81.6	199.7	366.4	> 600
AO2K	31.7	78.1	149.5	258.2
AO2M	12.7	27.3	49.7	79.4
m	90	110	130	150
n	60	60	60	60
$\lvert\mathcal{P}(L)\rvert^*$	4213716	6880963	9796009	13314592
$\lvert H\rvert^*$	6.3	6.6	6.8	6.9

Таблица 3.7: Случайные матрицы. Случай $30 \leqslant m = n \leqslant 90$

ОПТ	0.015	0.16	1.5	10.4	66.7
MMCS	**0.006**	0.2	2.3	17.5	112.4
RUNC	0.009	0.12	1.2	8.9	56.5
RUNC-M	0.007	**0.1**	**1.**	**7.6**	**48.2**
RS	0.012	0.2	2.2	15.7	99.7
ρUNC	0.01	0.13	1.5	10.8	70.
AO1	0.09	2.9	40.7	337.8	> 600
AO1K	0.03	0.8	11.1	93.8	> 600
AO1M	0.016	0.25	3.	25.6	174.
AO2	0.05	1.2	15.8	130.7	> 600
AO2K	0.023	0.5	6.1	52.	369.5
AO2M	0.016	0.24	2.9	25.2	168.
m	30	45	60	75	90
n	30	45	60	75	90
$\lvert\mathcal{P}(L)\rvert^*$	8113	160566	1618553	10921285	64273167
$\lvert H\rvert^*$	4.8	5.5	5.9	6.2	6.5

вершин. Это обстоятельство наводит на мысль, что число лишних шагов слабо коррелирует со скоростью работы алгоритма.

Тестирование алгоритмов также производилось на модельных данных из [77]. Использовались матрицы инцидентности следующих гиперграфов:

1) $M(n)$ — граф сочетаний, содержащий n вершин и $m = n/2$ ребер вида $\{2i - 1, 2i\}, i \in \{1, \ldots, m\}$ (число неприводимых покрытий матрицы инцидентности гиперграфа $M(n)$ равно 2^m);

2) $DM(n)$ — двойственный к графу $M(n)$ гиперграф (гиперграф $\mathcal{H}^d$ называется двойственным к гиперграфу $\mathcal{H}$, если ребрами $\mathcal{H}^d$ являются минимальные вершинные покрытия $\mathcal{H}$);

Таблица 3.8: Коэффициент эффективности. Случайные матрицы. Случай $m = 10, 50 \leqslant n \leqslant 300$

ОПТ	61	68	72	74	76	77
RUNC	77	85	88	90	92	92
RUNC-M	**86**	**90**	**92**	**93**	**94**	**94**
PUNC	80	87	90	91	93	**94**
AO1	17	27	35	41	45	49
AO1K	45	65	74	79	83	85
AO1M	62	75	82	85	87	89
AO2	21	31	38	43	48	51
AO2K	48	66	74	79	83	85
AO2M	63	76	82	85	87	89
m	10	10	10	10	10	10
n	50	100	150	200	250	300
$\lvert\mathcal{P}(L)\rvert^*$	8361	146248	836107	3257326	8666765	20658771
$\lvert H\rvert^*$	3.8	4.1	4.3	4.5	4.6	4.7

Таблица 3.9: Коэффициент эффективности. Случайные матрицы. Случай $m = 20$, $50 < n < 150$

ОПТ	52	56	59	62	63
RUNC	70	76	79	82	83
RUNC-M	**81**	**84**	**86**	**88**	**89**
PUNC	75	81	84	86	87
AO1	5	7	9	11	13
AO1K	18	26	32	38	41
AO1M	36	46	52	57	61
AO2	8	11	13	15	17
AO2K	23	31	36	42	44
AO2M	40	49	54	58	62
m	20	20	20	20	20
n	50	75	100	125	150
$\lvert\mathcal{P}(L)\rvert^*$	44552	314154	1269495	4254740	10321112
$\lvert H\rvert^*$	4.6	4.8	4.9	5.	5.1

3) $SDFP(n)$ — самодвойственный гиперграф с $n = 7k+2$ вершинами и ребрами $\{\{n\}\cup E{:}E\in(FP(k))^d\}\cup\{\{n-1\}\cup E{:}E\in FP(k)\}\cup\{\{n-1,n\}\}$, где гиперграф $FP(1)=\{\{1,2,3\},\{1,5,6\},\{1,7,4\},\{2,4,5\},\{2,6,7\},\{3,4,6\},\{3,5,7\}\}$ задает проективную плоскость Фано, а гиперграф $F(k), k > 1$, имеет ребра вида $\{E\cup(E+7)\cup\ldots\cup(E+7(k-1)) : E\in FP(1)\}$ (через $E+b$ обозначается множество $\{e+b : e\in E\}$; гиперграф $\mathcal{H}$ называется самодвойственным, если $\mathcal{H}^d=\mathcal{H}$);

4) $TH(n)$ — пороговый граф, содержащий четное число вершин n и $m = \frac{n^2}{4}$ ребер вида $\{i, 2j\}, 1 \leqslant i < 2j \leqslant n$ (число неприводимых покрытий матрицы инцидентности гиперграфа $TH(n)$ равно $n/2+1$);

Таблица 3.10: Коэффициент эффективности. Случайные матрицы. Случай $m = 30$, $50 \leqslant n \leqslant 110$

ОПТ	48	52	54	56
RUNC	66	71	74	77
RUNC-M	**79**	**81**	**83**	**84**
PUNC	73	78	80	83
AO1	2	3	4	5
AO1K	8	11	14	18
AO1M	25	30	35	39
AO2	5	6	7	8
AO2K	13	17	20	23
AO2M	30	34	38	42
m	30	30	30	30
n	50	70	90	110
$\lvert\mathcal{P}(L)\rvert^*$	113307	608535	2772442	7917863
$\lvert H\rvert^*$	5.	5.1	5.3	5.4

Таблица 3.11: Коэффициент эффективности. Случайные матрицы. Случай $50 \leqslant m \leqslant 300$, $n = 40$

ОПТ	41	35	32	30	29	28
RUNC	56	49	44	41	40	38
RUNC-M	**73**	**69**	**66**	**65**	**63**	**62**
PUNC	65	61	58	56	55	53
AO1	1	1	1	–	–	–
AO1K	2	1	1	–	–	–
AO1M	11	5	3	2	2	2
AO2	2	1	1	1	1	1
AO2K	6	2	1	1	1	1
AO2M	17	10	7	6	5	4
m	50	100	150	200	250	300
n	40	40	40	40	40	40
$\lvert\mathcal{P}(L)\rvert^*$	94389	333177	712507	1155969	1636729	2183946
$\lvert H\rvert^*$	5.5	6.3	6.8	7.1	7.4	7.6

5) $SDTH(n)$ — самодвойственный пороговый гиперграф с n вершинами и $m{=}\frac{(n-2)^2}{4}+\frac{n}{2}+1$ ребрами вида $\{\{n\}\cup E : E \in (TH(n-2))^d\}\cup\{\{n-1\}\cup E : E \in TH(n-2)\}\cup\{\{n-1,n\}\}$.

Тестирование алгоритмов также проводилось на прикладных задачах из [77]. Использовались матрицы инцидентности гиперграфов, возникающих в следующих прикладных задачах.

1) На основании данных о сделанных игроками ходах в игре Connect-4 составлены два гиперграфа: WIN и $LOSE$. Данные взяты из UCI Machine Learning Repository [82]. Каждое ребро гиперграфа WIN ($LOSE$) описывает состояние первого игрока при его выигрыше (проигрыше). Минимальное вершинное покрытие WIN ($LOSE$) определяет оптимальный путь, приводящий первого игрока к выигрышу (проигрышу). Гиперграфы $WIN(m)$ и $LOSE(m)$ состоят из первых m ребер гиперграфов WIN и $LOSE$ соответственно.

Таблица 3.12: Коэффициент эффективности. Случайные матрицы. Случай $70 \leqslant m \leqslant 230$, $n = 50$

ОПТ	40	37	34	32	31
RUNC	56	52	49	47	44
RUNC-M	**74**	**70**	**68**	**67**	**66**
PUNC	67	65	63	62	60
AO1	1	1	–	–	–
AO1K	1	1	1	–	–
AO1M	9	5	4	3	2
AO2	2	1	1	1	–
AO2K	4	2	2	1	1
AO2M	14	10	8	6	6
m	70	110	150	190	230
n	50	50	50	50	50
$\lvert\mathcal{P}(L)\rvert^*$	705711	1813930	3593400	5682139	8450430
$\lvert H\rvert^*$	6.	6.5	6.9	7.1	7.4

Таблица 3.13: Коэффициент эффективности. Случайные матрицы. Случай $90 \leqslant m \leqslant 150$, $n = 60$

ОПТ	39	38	37	36
RUNC	57	55	53	52
RUNC-M	**73**	**72**	**71**	**70**
PUNC	69	68	67	67
AO1	1	–	–	–
AO1K	1	1	–	–
AO1M	8	6	5	4
AO2	1	1	1	1
AO2K	3	2	2	2
AO2M	12	10	9	8
m	90	110	130	150
n	60	60	60	60
$\lvert\mathcal{P}(L)\rvert^*$	4213716	6880963	9796009	13314592
$\lvert H\rvert^*$	6.3	6.6	6.8	6.9

2) Гиперграфы BMS и ACC сформированы на основе проблемы поиска совместно встречающихся наборов. Использовались прикладные задачи «BMS-WebView-2» и «accidents» из FIMI Repository [83]. Гиперграфы $BMS(m)$ и $ACC(m)$ состоят из первых m ребер гиперграфов BMS и ACC соответственно. Особенностью этих задач является то, что в каждом ребре гиперграфа содержатся почти все его вершины.

Результаты счёта на модельных данных приведены в таблицах. 3.15–3.19. Результаты счёта на прикладных задачах приведены в таблицах. 3.20–3.23. В названиях таблиц 3.15–3.23 указаны типы задач. Столбцы таблиц 3.15–3.23 (кроме первого) соответствуют конкретным задачам, параметры которых приведены в нижних строках таблиц. В строках «m» и «n» указаны размеры матриц. В строках «$|\mathcal{P}(L)|$» и «$|H|^*$» указаны соответственно число неприводимых покрытий и средняя длина неприводимого покрытия матрицы. Остальные строки таблицы содержат

Таблица 3.14: Коэффициент эффективности. Случайные матрицы. Случай $30 \leqslant m = n \leqslant 90$

ОПТ	43	42	43	44	43
RUNC	56	59	61	62	63
RUNC-M	**74**	**75**	**76**	**77**	**77**
PUNC	63	68	71	73	75
AO1	1	1	1	1	–
AO1K	4	3	2	2	–
AO1M	17	14	12	11	10
AO2	4	3	2	2	–
AO2K	9	7	5	5	4
AO2M	22	20	17	15	14
m	30	45	60	75	90
n	30	45	60	75	90
$\lvert\mathcal{P}(L)\rvert^*$	8113	160566	1618553	10921285	64273167
$\lvert H\rvert^*$	4.8	5.5	5.9	6.2	6.5

Таблица 3.15: Модельные данные. Граф сочетаний $M(n)$

ОПТ	< 0.001	0.031	0.05	0.09	0.17	0.3	0.7	1.4	2.9	5.8
MMCS	0.016	0.05	0.11	0.16	0.3	0.7	1.6	3.1	6.3	13.2
RUNC	< 0.001	**0.016**	**0.016**	**0.05**	0.12	0.23	**0.5**	1.1	2.3	4.6
RUNC-M	< 0.001	**0.016**	0.031	0.06	0.14	0.27	0.6	1.1	2.4	7.
RS	0.05	0.06	0.11	0.2	0.5	0.9	1.7	4.2	8.6	16.8
PUNC	0.016	0.031	0.06	0.17	0.31	0.6	1.4	2.8	5.8	12.2
AO1	0.6	1.9	5.8	17.5	48.1	164.2	446.4	> 600	> 600	> 600
AO1K	0.016	**0.016**	0.031	0.06	**0.11**	**0.22**	**0.5**	**1.**	**1.9**	**4.2**
AO1M	0.016	**0.016**	0.05	0.06	**0.11**	0.25	**0.5**	**1.**	2.1	5.3
AO2	0.6	1.7	5.7	16.1	48.	154.5	460.7	> 600	> 600	> 600
AO2K	< 0.001	**0.016**	**0.016**	**0.05**	0.12	0.23	0.6	**1.**	2.3	4.7
AO2M	0.016	**0.016**	0.031	0.06	0.12	0.25	0.6	**1.**	2.3	5.1
m	14	15	16	17	18	19	20	21	22	23
n	29	31	33	35	37	39	41	43	45	47
$\lvert\mathcal{P}(L)\rvert$	16384	32768	65536	131072	262144	524288	1048576	2097152	4194304	8388608
$\lvert H\rvert^*$	14.	15.	16.	17.	18.	19.	20.	21.	22.	23.

время работы в секундах каждого из тестируемых алгоритмов. В случае, когда алгоритм в течении 600 секунд не заканчивал работу, выполнялась принудительная остановка программы, и в таблицу с результатами заносилась отметка «> 600».

Результаты счёта на модельных данных также, как и на случайных матрицах, показывают, что среди сравниваемых алгоритмов нет абсолютного лидера.

На матрице инцидентности графа сочетаний $M(n)$ лучшие результаты демонстрирует алгоритм RUNC, от которого незначительно отстают алгоритмы дуализации ОПТ и RUNC-M. По всей видимости, скорость работы достигается за счёт «удаления запрещённых столбцов», которое является общим для этих алгоритмов. Заметим, что изначально в матрице L вес каждой строки ранен 2. Поэтому «выбрасывание» охватывающих строк алгоритмом ОПТ и поиск строки с минимальным

Таблица 3.16: Модельные данные. Двойственный к $M(n)$ гиперграф $DM(n)$

ОПТ	0.06	0.8	11.2	46.9	184.5	> 600	> 600	> 600	> 600
MMCS	< 0.001	0.11	0.9	3.1	10.	35.	122.3	399.9	> 600
RUNC	0.016	0.05	0.27	0.8	2.5	8.9	28.7	93.6	569.9
RUNC-M	< 0.001	**0.031**	0.28	0.8	2.7	9.5	31.7	109.2	**309.2**
RS	0.016	0.06	0.6	2.2	6.	20.3	70.	283.6	> 600
PUNC	0.016	**0.031**	**0.17**	**0.5**	**1.6**	**5.6**	**17.3**	**51.7**	315.4
AO1	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO1K	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO1M	0.7	44.4	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO2	5.	438.8	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO2K	1.9	179.4	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO2M	0.28	10.3	364.2	> 600	> 600	> 600	> 600	> 600	> 600
m	1024	4096	16384	32768	65536	131072	262144	524288	1048576
n	21	25	29	31	33	35	37	39	41
$\|\mathcal{P}(L)\|$	10	12	14	15	16	17	18	19	20
$\|H\|^*$	2.	2.	2.	2.	2.	2.	2.	2.	2.

Таблица 3.17: Модельные данные. Самодвойственный гиперграф $SDPF(n)$

ОПТ	< 0.001	< 0.001	0.05	2.7	200.9	> 600
MMCS	< 0.001	0.016	**0.016**	0.5	25.3	> 600
RUNC	< 0.001	< 0.001	**0.016**	0.5	20.4	> 600
RUNC-M	< 0.001	< 0.001	**0.016**	0.3	**9.1**	440.2
RS	< 0.001	< 0.001	**0.016**	0.4	15.	> 600
PUNC	0.016	0.016	**0.016**	**0.25**	10.4	**329.5**
AO1	0.016	1.	> 600	> 600	> 600	> 600
AO1K	< 0.001	0.05	> 600	> 600	> 600	> 600
AO1M	< 0.001	0.016	0.031	7.4	> 600	> 600
AO2	< 0.001	0.031	24.7	> 600	> 600	> 600
AO2K	< 0.001	< 0.001	1.7	> 600	> 600	> 600
AO2M	< 0.001	0.016	0.19	36.4	> 600	> 600
m	15	64	365	2430	16843	117692
n	10	17	24	31	38	45
$\|\mathcal{P}(L)\|$	15	64	365	2430	16843	117692
$\|H\|^*$	3.9	6.3	9.6	12.9	16.	19.

весом алгоритмом RUNC-M требуют лишних вычислительных затрат, но не дают сокращения числа вершин дерева решений.

На матрице инцидентности гиперграфа $DM(n)$ лучшие результаты у алгоритма PUNC, который наименее чувствителен к росту числа строк входной матрицы. Напомним, что гиперграф $DM(n)$ имеет $m = 2^{\frac{n}{2}}$ ребер, просмотр которых на каждой итерации алгоритма требует довольно много времени уже при достаточно малых n. Наиболее критичен к числу строк входной матрицы алгоритм ОПТ, теоретическая сложность шага которого пропорциональна m^2.

Дуализацию порогового графа $TH(n)$ и самодвойственного гиперграфа $SDTH(n)$ быстрее других выполняют алгоритмы RS и MMCS. Матрицы инцидентности $TH(n)$ и $SDTH(n)$ являются сильно разреженными, что по словам

Таблица 3.18: Модельные данные. Пороговый граф $TH(n)$

ОПТ	0.016	0.14	1.	4.9	16.5	42.5	100.2	214.5	412.7
MMCS	0.031	0.031	0.016	< 0.001	< 0.001	**0.016**	**0.016**	**0.05**	**0.05**
RUNC	< 0.001	< 0.001	0.016	0.016	0.031	0.05	0.08	0.12	0.19
RUNC-M	< 0.001	0.016	< 0.001	0.016	0.05	0.05	0.08	0.14	0.2
RS	< 0.001	0.016	0.016	0.016	< 0.001	**0.016**	**0.016**	**0.05**	0.06
ρUNC	< 0.001	0.016	0.016	0.05	0.08	0.14	0.19	0.31	0.5
AO1	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO1K	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO1M	0.016	0.09	0.3	1.	2.5	5.2	9.8	16.9	30.9
AO2	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO2K	103.6	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO2M	0.2	2.8	22.3	104.6	364.4	> 600	> 600	> 600	> 600
m	400	900	1600	2500	3600	4900	6400	8100	10000
n	41	61	81	101	121	141	161	181	201
$\lvert\mathcal{P}(L)\rvert$	21	31	41	51	61	71	81	91	101
$\lvert H\rvert^*$	29.	44.	59.	74.	89.	104.	119.	134.	149.

Таблица 3.19: Модельные данные. Самодвойственный гиперграф $SDTH(n)$

ОПТ	4.8	15.8	42.2	103.2	217.8	411.1	> 600	> 600	> 600	> 600	> 600
MMCS	**0.016**	**0.031**	0.031	**0.05**	0.08	**0.09**	**0.14**	**0.23**	0.4	**0.5**	0.7
RUNC	**0.016**	**0.031**	0.05	0.08	0.11	0.19	0.4	0.7	1.4	2.	3.1
RUNC-M	0.031	**0.031**	0.05	0.09	0.12	0.23	0.4	0.9	1.3	2.	3.2
RS	**0.016**	**0.031**	**0.016**	**0.05**	**0.06**	**0.09**	0.16	**0.23**	**0.3**	0.6	**0.6**
ρUNC	0.08	0.17	0.3	0.5	0.7	1.7	2.2	5.7	20.9	12.7	27.6
AO1	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO1K	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO1M	2.4	6.	13.6	28.	52.6	85.6	172.	402.9	> 600	> 600	> 600
AO2	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO2K	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO2M	112.9	404.4	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600
m	2552	3662	4972	6482	8192	10102	14522	19742	25762	32582	40202
n	103	123	143	163	183	203	243	283	323	363	403
$\lvert\mathcal{P}(L)\rvert$	2552	3662	4972	6482	8192	10102	14522	19742	25762	32582	40202
$\lvert H\rvert^*$	4.4	4.4	4.5	4.5	4.5	4.5	4.5	4.5	4.5	4.5	4.5

Таблица 3.20: Прикладные задачи. Гиперграф $WIN(m)$

ОПТ	0.016	0.05	0.5	1.5	27.8	517.7	> 600	> 600	> 600	> 600
MMCS	< 0.001	< 0.001	**0.016**	0.06	0.3	2.9	12.	50.5	212.8	> 600
RUNC	< 0.001	< 0.001	**0.016**	0.05	0.23	2.3	13.	57.9	511.	> 600
RUNC-M	< 0.001	< 0.001	0.031	**0.031**	**0.17**	**1.3**	**5.1**	**20.3**	**75.5**	**306.5**
RS	< 0.001	0.016	**0.016**	0.08	0.5	5.3	21.4	84.7	> 600	> 600
ρUNC	< 0.001	0.031	0.08	0.22	1.7	20.2	104.	530.9	> 600	> 600
AO1	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO1K	2.2	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO1M	0.016	0.19	5.9	29.2	465.	> 600	> 600	> 600	> 600	> 600
AO2	1.3	35.4	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO2K	0.031	0.7	20.8	79.7	> 600	> 600	> 600	> 600	> 600	> 600
AO2M	0.016	0.17	3.2	10.6	150.7	> 600	> 600	> 600	> 600	> 600
m	100	200	400	800	1600	3200	6400	12800	25600	44473
n	76	78	78	78	80	82	82	84	84	85
$\lvert\mathcal{P}(L)\rvert$	287	1145	6069	11675	71840	459502	1277933	4587967	11614885	31111249
$\lvert H\rvert^*$	10.8	12.1	14.3	15.1	16.7	17.9	18.9	19.9	20.8	21.9

Таблица 3.21: Прикладные задачи. Гиперграф $LOSE(m)$

ОПТ	0.031	0.22	1.3	4.2	66.5	> 600	> 600	> 600	> 600
MMCS	< 0.001	0.05	0.11	0.3	1.1	12.9	41.8	185.8	519.5
RUNC	0.016	0.031	0.09	0.5	2.2	10.3	58.5	278.6	> 600
RUNC-M	< 0.001	**0.016**	**0.05**	**0.17**	**0.5**	**4.3**	**12.5**	**49.2**	**137.7**
RS	0.016	0.06	0.16	0.5	1.7	17.1	107.1	449.7	> 600
PUNC	0.016	0.11	0.4	1.5	5.8	71.1	501.2	> 600	> 600
AO1	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO1K	7.1	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO1M	0.016	0.23	4.2	61.5	> 600	> 600	> 600	> 600	> 600
AO2	2.3	116.9	> 600	> 600	> 600	> 600	> 600	> 600	> 600
AO2K	0.031	0.9	33.9	> 600	> 600	> 600	> 600	> 600	> 600
AO2M	0.031	0.23	2.7	28.7	325.9	> 600	> 600	> 600	> 600
m	100	200	400	800	1600	3200	6400	12800	16635
n	77	77	79	81	81	81	81	85	85
$\|\mathcal{P}(L)\|$	2341	22760	33087	79632	212761	2396735	4707877	16405082	39180611
$\|H\|^*$	11.2	12.5	13.7	14.8	15.9	17.2	17.6	19.3	20.1

Таблица 3.22: Прикладные задачи. Гиперграф $ACC(m)$

ОПТ	< 0.001	< 0.001	0.016	0.05	0.23	1.1	8.2	136.3
MMCS	< 0.001	0.016	0.016	0.06	0.27	0.8	3.7	33.
RUNC	0.016	0.031	0.016	0.031	0.09	**0.25**	**0.9**	7.6
RUNC-M	< 0.001	< 0.001	< 0.001	**0.016**	**0.08**	**0.25**	**0.9**	**7.3**
RS	< 0.001	< 0.001	0.016	0.05	0.3	0.8	3.4	35.3
PUNC	< 0.001	< 0.001	0.016	0.031	0.27	1.4	9.9	208.3
AO1	0.031	25.3	> 600	> 600	> 600	> 600	> 600	> 600
AO1K	0.016	0.3	9.5	300.	> 600	> 600	> 600	> 600
AO1M	< 0.001	0.12	0.9	5.3	76.3	> 600	> 600	> 600
AO2	< 0.001	0.11	0.8	5.4	68.7	> 600	> 600	> 600
AO2K	< 0.001	0.031	0.3	1.5	23.8	196.2	> 600	> 600
AO2M	< 0.001	0.05	0.3	1.7	23.1	175.6	> 600	> 600
m	81	447	990	2000	4322	10968	32207	135439
n	64	64	81	81	336	336	336	442
$\|\mathcal{P}(L)\|$	253	1039	1916	3547	7617	17486	47137	185218
$\|H\|^*$	2.6	3.8	4.2	4.7	5.1	5.7	6.5	7.3

авторов алгоритмов RS и MMCS является одним из условий эффективности работы этих алгоритмов.

С задачей дуализации самодвойственого графа $SDFP(n)$ снова лучше всех справился алгоритм PUNC, от которого в основном незначительно отстает алгоритм дуализации RUNC-M. Тестирование на прикладных задачах показывает, что лучшим является алгоритм RUNC-M, преимущество которого особенно очевидно на входных матрицах большого размера.

Таблица 3.23: Прикладные задачи. Гиперграф $BMS(m)$

ОПТ	**0.031**	**0.16**	**0.7**	3.9	**21.1**	105.3	310.6
MMCS	**0.031**	0.17	1.3	15.7	117.4	510.8	> 600
RUNC	0.05	0.19	0.8	**3.8**	24.7	**94.2**	**261.7**
RUNC-M	0.06	0.2	0.8	4.	25.1	113.6	315.6
RS	0.06	**0.16**	1.3	14.2	98.2	446.7	> 600
PUNC	**0.031**	0.3	3.5	34.8	217.4	> 600	> 600
AO1	0.09	0.8	13.5	> 600	> 600	> 600	> 600
AO1K	0.09	0.8	9.8	> 600	> 600	> 600	> 600
AO1M	0.12	1.2	20.5	306.4	> 600	> 600	> 600
AO2	0.11	0.8	8.9	116.8	> 600	> 600	> 600
AO2K	0.09	0.8	8.9	106.3	> 600	> 600	> 600
AO2M	0.11	1.2	18.3	286.4	> 600	> 600	> 600
m	62	237	823	2591	6946	17315	30405
n	3340	3340	3340	3340	3340	3340	3340
$\lvert\mathcal{P}(L)\rvert$	4616	15993	89448	438867	1289303	2297560	3064937
$\lvert H\rvert^*$	1.3	1.8	2.	2.	2.	2.	2.1

Заключение

1. Предложена общая схема синтеза логических корректоров, голосующих по корректным предикатам. Показано, что схема логического корректора общего вида может быть использована для описания классических логических алгоритмов распознавания и ранее построенных логических корректоров.

2. Построен практический логический корректор POLAR с поляризуемой корректирующей функцией.

3. Разработана методика повышения качества распознавания и скорости обучения логических корректоров, основанная на построении локальных базисов классов и итеративном формировании семейств голосующих предикатов по принципу бустинга.

4. Построены новые асимптотически оптимальные алгоритмы дуализации AO1M, AO1K, AO2M, AO2K, RUNC, RUNC-M, PUNC, в которых снижение времени счёта достигается за счёт уменьшения общего числа вершин дерева решений. Показано, что построенные алгоритмы достаточно быстро обрабатывают булевы матрицы большого размера.

Одним из дальнейших направлений исследований видится обобщение методов алгебро-логической коррекции на случай, когда в задаче распознавания на множествах значений признаков определены отношения частичного порядка. Практический интерес представляют частичные порядки, являющиеся цепями, антицепями, полурешётками, решётками или лесами. При выполнении коррекции потребуются эффективные перечислительные алгоритмы, для решения задач, обобщающих дуализацию.

Список литературы

1. Журавлев, Ю. И. О математических принципах классификации предметов и явлений / Ю. И. Журавлев, А. Н. Дмитриев, Ф. П. Кренделев // Дискретный анализ, Сб. научн. тр. — Т. 7. — Ин-т математики СО АН СССР Новосибирск, 1966. — С. 3–15.
2. Журавлёв, Ю. И. Алгоритмы распознавания, основанные на вычислении оценок / Ю. И. Журавлёв, В. В. Никифоров // Кибернетика. — 1971. — № 3.
3. Вайнцвайг, М. Н. Алгоритм обучения распознаванию образов «Кора» / М. Н. Вайнцвайг; Под ред. В. Н. Вапник. — М.: Советское радио, 1973. — С. 110–116.
4. Дюкова, Е. В. Поиск информативных фрагментов описаний объектов в дискретных процедурах распознавания / Е. В. Дюкова, Н. В. Песков // ЖВМиМФ. — 2002. — Т. 42, № 5. — С. 741–753. — www.ccas.ru/frc/papers/djukova02poisk.pdf.
5. Баскакова, Л. В. Модель распознающих алгоритмов с представительными наборами и системами опорных множеств / Л. В. Баскакова, Ю. И. Журавлев // ЖВМ и МФ. — 1981. — Т. 21, № 5. — С. 1264–1275.
6. Дюкова, Е. В. Построение распознающих процедур на базе элементарных классификаторов / Е. В. Дюкова, Н. В. Песков // Математические вопросы кибернетики / Под ред. О. Б. Лупанов. — М.: Физматлит, 2005. — Т. 14. — www.ccas.ru/frc/papers/djukova05construction.pdf.
7. Дюкова, Е. В. Дискретные (логические) процедуры распознавания: принципы конструирования, сложность реализации и основные модели. / Е. В. Дюкова. — М.: Прометей, 2003. — 29 с. — Учебное пособие для студентов математических факультетов педвузов. www.ccas.ru/frc/papers/djukova03mp.pdf.
8. Djukova, E. V. Selection of typical objects in classes for recognition problems / E. V. Djukova, N. V. Peskov // Pattern Recognition and Image Analysis. — 2002. — Vol. 12, no. 3. — P. 243–249. — www.ccas.ru/frc/papers/djukova02selection.pdf.
9. Журавлёв, Ю. И. Об алгоритмах распознавания с представительными наборами (о логических алгоритмах) / Ю. И. Журавлёв // ЖВМиМФ. — 2002. — Т. 42, № 9. — С. 1425–1435. — www.ccas.ru/frc/papers/zhuravlev02jvm.pdf.
10. Рязанов, В. В. О некоторых моделях голосования и методах их оптимизации / В. В. Рязанов, О. В. Сенько // Распознавание, классификация, прогноз. — 1990. — Т. 3. — С. 106–145.
11. Рязанов, В. В. Логические закономерности в задачах распознавания (параметрический подход) / В. В. Рязанов // Ж. вычисл. матем. и матем. физ. — 2007. — Т. 47, № 10. — С. 1793–1808.
12. Ковшов, Н. В. Алгоритмы поиска логических закономерностей в задачах распознавания / Н. В. Ковшов, В. Л. Моисеев, В. В. Рязанов // Ж. вычисл. матем. и матем. физ. — 2008. — Т. 48, № 2. — С. 329–344.

13. Обработка вещественнозначной информации логическими процедурами распознавания / Е В Дюкова, Ю И Журавлев, Н В Песков, А А Сахаров // Искуственный интеллект. НАН Украины. — 2004. — № 2. — С. 80–85.

14. Increasing the efficiency of combinatorial logical data analysis in recognition and classification problems / E. V. Djukova, A. S. Inyakin, N. V. Peskov, A. A. Sakharov // Pattern Recognition and Image Analysis. — 2005. — Vol. 16, no. 4. — P. 707–711. — www.ccas.ru/frc/papers/djukova06increasing.pdf.

15. Дюкова, Е. В. Об алгебро-логиЧеском синтезе корректных процедур распознавания на базе элементарных алгоритмов / Е. В. Дюкова, Ю. И. Журавлев, К. В. Рудаков // ЖВМ и МФ. — 1996. — Т. 36, № 8. — С. 216–223.

16. Dyukova, E. V. Models of recognition procedures with logical correctors / E. V. Dyukova, P. A. Prokofjev // Pattern Recognition and Image Analysis. — 2013. — Vol. 23, no. 2. — P. 235–244.

17. Дюкова, Е. В. Методы обучения логических процедур распознавания, основанных на семействах корректных наборов элементарных классификаторов / Е. В. Дюкова, П. А. Прокофьев // Интеллектуализация обработки информации: 9-я Международная конференция. Черногория, г. Будва, 2012: Сборник докладов. — М.: Торус Пресс, 2012. — С. 67–70.

18. Djukova, E. V. Construction of an ensemble of logical correctors on the basis of elementary classifiers / E V Djukova, Yu I Zhuravlev, R M Sotnezov // Pattern Recognition and Image Analysis. — 2011. — Vol. 21, no. 4. — P. 599–605.

19. Дюкова, Е. В. Об алгебро-логической коррекции в задачах распознавания по прецедентам / Е. В. Дюкова, М. М. Любимцева, П. А. Прокофьев // Машинное обучение и анализ данных. — 2013. — Т. 1, № 6. — С. 705–713. — http://jmlda.org/papers/doc/2013/no6/Djukova2013ALCorrection.pdf.

20. Журавлёв, Ю. И. Об алгебраических методах в задачах распознавания и классификации / Ю. И. Журавлёв // Распознавание, классификация, прогноз. — 1988. — Т. 1. — С. 9–16. — www.ccas.ru/frc/papers/zhuravlev88rkp.pdf.

21. Журавлёв, Ю. И. Об алгебраическом подходе к решению задач распознавания или классификации / Ю. И. Журавлёв // Проблемы кибернетики. — 1978. — Т. 33. — С. 5–68. — www.ccas.ru/frc/papers/zhuravlev78prob33.pdf.

22. Журавлёв, Ю. И. Об алгебраической коррекции процедур обработки (преобразования) информации / Ю. И. Журавлёв, К. В. Рудаков // Проблемы прикладной математики и информатики. — 1987. — С. 187–198. — www.ccas.ru/frc/papers/zhurrud87correct.pdf.

23. Воронцов, К. В. Оптимизационные методы линейной и монотонной коррекции в алгебраическом подходе к проблеме распознавания / К. В. Воронцов // ЖВМ и МФ. — 2000. — Т. 40, № 1. — С. 166–176. — www.ccas.ru/frc/papers/voron00jvm.pdf.

24. Воронцов, К. В. Проблемно-ориентированные методы алгебраического подхода. — 2002. — www.ccas.ru/frc/papers/voron02po4.pdf.

25. Рудаков, К. В. Универсальные и локальные ограничения в проблеме коррекции эвристических алгоритмов / К. В. Рудаков // Кибернетика. — 1987. — № 2. — С. 30–35. — www.ccas.ru/frc/papers/rudakov87universal.pdf.

26. Любимцева, М. М. Логические корректоры в задачах распознавания / М. М. Любимцева // Сборник тезисов лучших дипломных работ факультета ВМК МГУ 2014 года — М: МАКС ПРЕСС. — 2014. — С. 47–49.
27. Johnson, D. On generating all maximal independent sets / D.S. Johnson, M. Yannakakis, C.H. Papadimitriou // Information Processing Letters. — 1988. — Vol. 27, no. 3. — P. 119–123.
28. An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals and its application in joint generation / Leonid Khachiyan, Endre Boros, Khaled Elbassioni, Vladimir Gurvich // Discrete Applied Mathematics. — 2006. — Vol. 154, no. 16. — P. 2350–2372.
29. An efficient incremental algorithm for generating all maximal independent sets in hypergraphs of bounded dimension / E. Boros, V. Gurvich, K. Elbassioni, L. Khachiyan // Parallel Processing Letters. — 2000. — Vol. 10, no. 4. — P. 253–266.
30. Generating maximal independent sets for hypergraphs with bounded edge-intersections / E. Boros, K.M. Elbassioni, V. Gurvich, L. Khachiyan // in: M. Farach-Colton (Ed.), Proceedings of the 6th Latin American Symposium on Theoretical Informatics (LATIN 2004), Lecture Notes in Computer Science, Buenos Aires, Argentina. — 2004. — Vol. 2976. — P. 488–498.
31. Fredman, M. L. On the complexity of dualization of monotone disjunctive normal forms / Michael L Fredman, Leonid Khachiyan // Journal of Algorithms. — 1996. — Vol. 21, no. 3. — P. 618–628.
32. Djukova, E. V. Discrete recognition procedures: The complexity of realization / E. V. Djukova // Pattern Recognition and Image Analysis. — 2003. — Vol. 13, no. 1. — P. 8–10. — www.ccas.ru/frc/papers/djukova03discrete.pdf.
33. Djukova, E. V. On the complexity of discrete generation problems / E. V. Djukova, R. M. Sotnezov // Doklady Mathematics. — 2010. — Vol. 82, no. 3. — P. 847–849.
34. Djukova, E. V. Discrete methods of information analysis in recognition and algorithms synthesis / E. V. Djukova, Ju. I. Zhuravlev // Pattern Recognition and Image Analysis. — 1997. — Vol. 7, no. 2. — P. 192–207. — www.ccas.ru/frc/papers/djukova97discrete.pdf.
35. Дюкова, Е. В. Об асимптотически оптимальном алгоритме построения тупиковых тестов / Е В Дюкова // ДАН СССР. — 1997. — Т. 233, № 4. — С. 527–530.
36. Дюкова, Е. О сложности реализации некоторых процедур распознавания / Е.В. Дюкова // Журнал вычислительной математики и математической физики. — 1987. — Т. 27, № 1. — С. 114–127.
37. Дюкова, Е. В. О сложности реализации дискретных (логических) процедур распознавания / Елена Всеволодовна Дюкова // Журнал вычислительной математики и математической физики. — 2004. — Т. 44, № 3. — С. 562–572.
38. Дюкова, Е. В. Дискретный анализ признаковых описаний в задачах распознавания большой размерности / Елена Всеволодовна Дюкова, Юрий Иванович Журавлев // Журнал вычислительной математики и математической физики. — 2000. — Т. 40, № 8. — С. 1264–1278.
39. Дюкова, Е. В. Асимптотически оптимальное построение тупиковых покрытий целочисленной матрицы / Е В Дюкова, А С Инякин // Математические вопросы кибернетики. — 2008. — Т. 17. — С. 235–246.
40. Дюкова, Е. В. О сложности задачи дуализации / Е В Дюкова, Р М Сотнезов // Ж. вычисл. матем. и матем. физ. — 2012. — Т. 52, № 10. — С. 1926–1935.

41. Boosting the margin: a new explanation for the effectiveness of voting methods / Robert E. Schapire, Yoav Freund, Wee Sun Lee, Peter Bartlett // Annals of Statistics. — 1998. — Vol. 26, no. 5. — P. 1651–1686. — citeseer.ist.psu.edu/article/schapire98boosting.html.

42. Schapire, R. E. Improved boosting using confidence-rated predictions / Robert E. Schapire, Yoram Singer // Machine Learning. — 1999. — Vol. 37, no. 3. — P. 297–336. — citeseer.ist.psu.edu/article/singer99improved.html.

43. Воронцов, К. В. О проблемно-ориентированной оптимизации базисов задач распознавания / К. В. Воронцов // ЖВМ и МФ. — 1998. — Т. 38, № 5. — С. 870–880. — www.ccas.ru/frc/papers/voron98jvm.pdf.

44. Воронцов, К. В. Локальные базисы в алгебраическом подходе к проблеме распознавания. — Автореферат диссертации на соискание учёной степени к.ф.-м.н., М.: ВЦ РАН. — 1999. — www.ccas.ru/frc/thesis/VoronCanAutoref.pdf.

45. Peleg, D. Approximation algorithms for the label-cover max and red-blue set cover problems / David Peleg // Journal of Discrete Algorithms. — 2007. — Vol. 5, no. 1. — P. 55–64.

46. Miettinen, P. On the positive–negative partial set cover problem / Pauli Miettinen // Information Processing Letters. — 2008. — Vol. 108, no. 4. — P. 219–221.

47. Васильев, В. Г. Извлечение информации из текста с автоматическим построением правил / В. Г. Васильев, П. А. Прокофьев // Электронные библиотеки: перспективные методы и технологии, электронные коллекции. XIII Всероссийская научная конференция RCDL'2011. Воронеж, 19–22 октября 2011 г.: труды конференции. — Воронеж: Издательско-полиграфический центр Воронежского государственного университета, 2011. — С. 358–364.

48. Прокофьев, П. А. Дискретный подход при извлечении информации из текста с автоматическим построением правил (текстовых запросов) / П. А. Прокофьев // Математические методы распознавания образов: 15-я Всероссийская конференция, г. Петрозаводск, 11–17 сентября 2011 г.: Сборник докладов. — М.: МАКС Пресс, 2011. — С. 585–588.

49. Прокофьев, П. А. Классификация фрагментов текстов с описанием зависимостей правилами на интерпретируемом экспертами языке / П. А. Прокофьев // Вестник ВГУ, серия: Системный анализ и информационные технологии. — 2012. — № 1. — С. 174–178.

50. Djukova, E. V. Logical correctors in recognition problems / E. V. Djukova, M. M. Lyubimtseva, P. A. Prokofjev // 11th International Conference on Pattern Recognition and Image Analysis: New Information Technologies (PRIA-11-2013). Samara, September 23-28, 2013. — Vol. 1. — Samara: IPSI RAS, 2013. — P. 82–83.

51. Дюкова, Е. В. Логические корректора в задачах классификации по прецедентам / Е. В. Дюкова, М. М. Любимцева, П. А. Прокофьев // Математические методы распознавания образов: 16-я Всероссийская конференция, г. Казань, 6–12 сентября 2013 г.: Тезисы докладов. — М.: Торус Пресс, 2013. — С. 7.

52. Дюкова, Е. В. Вопросы эффективности логических корректоров / Е. В. Дюкова, Ю. И. Журавлёв, П. А. Прокофьев // Математические методы распознавания образов: Тезисы докладов 17-той Всероссийской конференции с международным участием, г. Светлогорск, 2015 г. — М.: Торус Пресс, 2015. — С. 70–71.

53. Дюкова, Е. В. Построение и исследование новых асимптотически оптимальных алгоритмов дуализации / Е. В. Дюкова, П. А. Прокофьев // Машинное обучение и анализ данных. — 2014. — Т. 1, № 8. — С. 1048–1067.

54. Djukova, E. V. Logical correctors in recognition / E. V. Djukova, M. M. Lyubimtseva, P. A. Prokofjev // Pattern Recognition and Image Analysis. — 2014. — Vol. 24, no. 3. — P. 358–364.

55. Дюкова, Е. В. Об асимптотически оптимальном перечислении неприводимых покрытий булевой матрицы / Е. В. Дюкова, П. А. Прокофьев // Прикладная дискретная математика. — 2014. — № 1. — С. 96–105.

56. Дюкова, Е. В. Новые асимптотически оптимальные алгоритмы дуализации / Е. В. Дюкова, П. А. Прокофьев // Интеллектуализация обработки информации: 10-я Международная конференция. Греция, о. Крит, 4–11 октября, 2012: Тезисы докладов. — М.: Торус Пресс, 2014. — С. 50–51.

57. Djukova, E. V. Asymptotically optimal dualization algorithms / E. V. Djukova, P. A. Prokofjev // Computational Mathematics and Mathematical Physics. — 2015. — Vol. 55, no. 5. — P. 891–905.

58. Дюкова, Е. В. Методы повышения эффективности логических корректоров / Е. В. Дюкова, Ю. И. Журавлёв, П. А. Прокофьев // Машинное обучение и анализ данных. — 2015. — Т. 1, № 11. — С. 1555–1583.

59. Ашуров, А. Р. Алгоритмы вычисления оценок для задачи распознавания объектов с континуальной начальной информацией / А. Р. Ашуров, К. В. Рудаков // ЖВМиМФ. — 1984. — Т. 24, № 12. — С. 1871–1880. — www.ccas.ru/frc/papers/rudakov84avo.pdf.

60. Dong, G. Caep: Classification by aggregating emerging patterns / Guozhu Dong, Xiuzhen Zhang, Limsoon Wong. — 1999. — P. 30–42.

61. Dong, G. Efficient mining of emerging patterns: Discovering trends and differences / Guozhu Dong, Jinyan Li. — 1999. — P. 43–52.

62. Fan, H. An efficient single-scan algorithm for mining essential jumping emerging patterns for classification / Hongjian Fan, Ramamohanarao Kotagiri // Advances in Knowledge Discovery and Data Mining. — Springer, 2002. — P. 456–462.

63. Журавлёв, Ю. И. Экстремальные задачи, возникающие при обосновании эвристических процедур / Ю. И. Журавлёв // Приблемы прикладной математики и механики. — М.: Наука, 1971. — С. 67–74. — www.ccas.ru/frc/papers/zhuravlev71extremal.pdf.

64. Logical analysis of numerical data / Endre Boros, Peter L. Hammer, Toshihide Ibaraki, Alexander Kogan // Mathematical Programming. — 2000. — Vol. 79. — P. 163–190.

65. Hammer, P. L. Logical analysis of data — an overview: From combinatorial optimization to medical applications / Peter L Hammer, Tibérius O Bonates // Annals of Operations Research. — 2006. — Vol. 148, no. 1. — P. 203–225.

66. Finding essential attributes from binary data / Endre Boros, Takashi Horiyama, Toshihide Ibaraki et al. // Annals of Mathematics and Artificial Intelligence. — 2003. — Vol. 39, no. 3. — P. 223–257.

67. Воронцов, К. В. Лекции по логическим алгоритмам классификации. — www.ccas.ru/voron/teaching.html. — 2007. — www.ccas.ru/voron/teaching.html.

68. On the red-blue set cover problem / Robert D Carr, Srinivas Doddi, Goran Konjevod, Madhav V Marathe // in: Proc. 11th ACM-SIAM Symp. on Discrete Algorithms. — 2000. — P. 345–353.

69. Meir, R. An introduction to boosting and leveraging / Ron Meir, Gunnar Ratsch // Advanced Lectures on Machine Learning, LNAI 2600. — 2003. — Р. 118–183.

70. Sotnezov, R. M. Genetic algorithms for problems of logical data analysis in discrete optimization and image recognition / R. M. Sotnezov // Pattern Recognition and Image Analysis. — 2009. — Vol. 19, no. 3. — Р. 469–477.

71. Eiter, T. New results on monotone dualization and generating hypergraph transversals / Thomas Eiter, Georg Gottlob, K. Makino // SIAM Journal on Computing. — 2003. — Vol. 32, no. 2. — Р. 514–537.

72. Elbassioni, K. Some fixed-parameter tractable classes of hypergraph duality and related problems / K. Elbassioni, M. Hagen, I. Rauf // In IWPEC. — 2008. — Р. 91–102.

73. Elbassioni, K. Output-sensitive algorithms for enumerating minimal transversals for some geometric hypergraphs / K. Elbassioni, K. Makino, I. Rauf // In ESA. — 2009.

74. Babin, M. Enumerating minimal hypotheses and dualizing monotone boolean functions on lattices / M. Babin, S.O. Kuznetsov // Formal Concept Analysis SE-5. — 2011. — Vol. 6628. — Р. 42–48.

75. Дюкова, Е. В. О решении систем булевых уравнений квазинельсоновского типа / Е В Дюкова // Вопросы кибернетики. — 1989. — С. 5–19.

76. Murakami, K. Efficient algorithms for dualizing large-scale hypergraphs / Keisuke Murakami, Takeaki Uno // Discrete Applied Mathematics. — 2014. — Vol. 170. — Р. 83–94.

77. Murakami, K. Efficient algorithms for dualizing large-scale hypergraphs / K. Murakami, T. Uno // Proceedings of the 15th Meeting on Algorithm Engineering and Experiments, ALENEX 2013, New Orleans, Louisiana, USA, 2013. SIAM. — 2013.

78. Bailey, J. A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns / James Bailey, Thomas Manoukian, Kotagiri Ramamohanarao // 2013 IEEE 13th International Conference on Data Mining / IEEE Computer Society. — 2003. — Р. 485–485.

79. Dong, G. Mining border descriptions of emerging patterns from dataset pairs / Guozhu Dong, Jinyan Li // Knowledge and Information Systems. — 2005. — Vol. 8, no. 2. — Р. 178–202.

80. Hébert, C. A data mining formalization to improve hypergraph minimal transversal computation / Céline Hébert, Alain Bretto, Bruno Crémilleux // Fundamenta Informaticae. — 2007. — Vol. 80, no. 4. — Р. 415–433.

81. Kavvadias, D. J. An efficient algorithm for the transversal hypergraph generation. / Dimitris J Kavvadias, Elias C Stavropoulos // J. Graph Algorithms Appl. — 2005. — Vol. 9, no. 2. — Р. 239–264.

82. UCI machine learning repository: Rep. / University of California, Irvine, School of Information and Computer Sciences; Executor: A. Asuncion, D.J. Newman: 2007. — www.ics.uci.edu/$\sim$mlearn/MLRepository.html.

83. Frequent itemset mining dataset repository. — http://fimi.ua.ac.be/data/.