Построение и исследование новых асимптотически оптимальных алгоритмов дуализации*

***Е. В. Дюкова*[1], *П. А. Прокофьев*[2]**
[1]edjukova@mail.ru; [2]p_prok@mail.ru
Вычислительный центр им. Дородницына РАН, Москва, ул. Вавилова, 42

В статье исследуется подход к синтезу эффективных в типичном случае (on average) алгоритмов для задачи дуализации, представленной в матричной формулировке (точный статус этой задачи в плане полиномиальной разрешимости не определен). Построены новые асимптотически оптимальные алгоритмы дуализации. Показано, что эти алгоритмы позволяют сократить временные затраты по сравнению с наиболее эффективными из известных алгоритмов дуализации.

Ключевые слова: дуализация; булева матрица; асимптотически оптимальный алгоритм; неприводимое покрытие; перечисление с полиномиальной задержкой

Construction and investigation of new asymptotically optimal algorithms for dualization*

E. V. Djukova, P. A. Prokofjev
Dorodnitsyn Computing Centre, Russian Academy of Sciences, 42 Vavilov Str., Moscow

Background: In this paper, an improvement and experimental justification of the asymptotically optimal approach to discrete enumeration problems is presented. The approach is aimed at effective on average algorithms construction. Dualization is considered as a fundamental enumeration problem. It is equivalent to irreducible coverings enumeration of a given boolean matrix. A column set H of a boolean matrix L is called an irreducible covering if (i) sub-matrix L^H formed by columns from H contains no zero-filled rows; and (ii) L^H contains each row from the list: $(1,0,0,\ldots,0,0),(0,1,0,\ldots,0,0),\ldots,(0,0,0,\ldots,0,1)$. Column set H satisfying the condition (ii) is called *compatible*.

Methods: Asymptotically optimal dualization algorithms enumerate with polynomial delay the "maximal" compatible column sets. Such algorithms construct a column set at each step. The solutions set is updated if the column set satisfies (i) and has not been found at previous steps. Otherwise, this step is considered "extra." The proportion of "extra" steps tends to zero for almost all boolean matrices of a given size as matrix size increases.

Results: Asymptotically optimal dualization algorithms RUNC, RUNC-M, and PUNC are suggested. These algorithms are shown to outperform the prior dualization algorithms.

Concluding Remarks: Despite the absence of estimates of the complexity of asymptotically optimal dualization algorithms for the worst case, asymptotically optimal approach is superior to other approaches, according to the experiments on real problems.

Keywords: dualization; Boolean matrix; asymptotically optimal algorithm; irreducible covering; polinomial delay enumerating

Рассматривается задача поиска неприводимых покрытий булевой матрицы. Сформулируем ее. Пусть $L = \|a_{ij}\|_{m\times n}$ — булевая матрица и пусть H — набор столбцов матрицы L.

Работа частично поддержана грантами РФФИ № 13-01-00787-а, № 14-07-00819-а и грантом президента РФ НШ-4908.2014.1.

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.

Через $\mathcal{P}(L)$ обозначается 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 дуализации, 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 сокращенную дизъюнктивную нормальную форму функции F.
2) Дан гиперграф $\mathcal{H}$ с n вершинами и m ребрами. Требуется найти все минимальные вершинные покрытия гиперграфа $\mathcal{H}$.

Эффективность алгоритмов перечисления принято оценивать сложностью шага [1]. 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 [1,2]. Например, в [1] построен алгоритм с задержкой $O(n^3)$ для случая, когда в каждой строке матрицы L не более двух единичных элементов, что в постановке 2) соответствует случаю: $\mathcal{H}$ — граф.

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 [3, 4] построен инкрементальный квазиполиномиальный алгоритм дуализации. В [5, 6] для ряда частных случаев дуализации построены инкрементальные полиномиальные алгоритмы.

Существует еще один подход к решению рассматриваемой задачи, основанный на понятии асимптотически оптимального алгоритма с полиномиальной задержкой. Подход впервые предложен в [7] 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 найде-

но 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 (см. [7–16]). В этих алгоритмах для построения $\mathcal{P}(L)$ 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 $(1,0,0,\ldots,0,0),(0,1,0,\ldots,0,0),\ldots,(0,0,0,\ldots,0,1)$, т.е. с точностью до перестановки строк содержит единичную подматрицу порядка r. Набор столбцов, удовлетворяющий условию 2), называется *совместимым*.

В асимптотически оптимальном алгоритме дуализации АО1 [7] в роли Z_1 выступает задача построения совокупности наборов столбцов матрицы L, удовлетворяющих 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, т.е. такой единичной подматрицей, которая не содержится в других единичных подматрицах. Максимальная единичная подматрица порождает максимальный совместимый набор столбцов, т.е. такой совместимый набор столбцов, который не содержится ни в каком другом совместимом наборе столбцов.

Схема работы алгоритма АО1 позволяет со сложностью шага $O(qmn)$, 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 $O(mn)$ 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 $O(mn)$ проверяет не был ли набор H построен на предыдущих шагах.

В алгоритме АО2 [12, 16], который является модификацией алгоритма АО1, с полиномиальной задержкой $O(qm^2n)$ 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 [16] на базе алгоритма АО2 были построены алгоритмы АО2К и АО2М, позволяющие сократить время счета.

Наименьшее число лишних шагов имеет асимптотически оптимальный алгоритм ОПТ [14], основанный на перечислении с полиномиальной задержкой $O(qm^2n)$ наборов столбцов матрицы L, удовлетворяющих условию 2) и некоторым дополнительным условиям, среди которых условие максимальности. Лишние шаги в алгоритме ОПТ возникают за счет построения максимальных совместимых наборов столбцов, не являющихся покрытиями, т.е. не удовлетворяющих условию 1).

Обоснование асимптотически оптимальных алгоритмов дуализации базируется на приведенных ниже теоремах 1 и 2. Пусть $\mathcal{S}(L)$ – множество всех единичных подматриц матрицы L.

Теорема 1. *Если* $\log m \leqslant (1-\varepsilon)\log n$, $0 < \varepsilon < 1$, *то для почти всех булевых матриц* L *размера* $m \times n$ *имеет место* $|\mathcal{S}(L)| \sim |\mathcal{P}(L)|, n \to \infty$.

Теорема 2. *Построение* $\mathcal{S}(L)$ *может быть осуществлено с полиномиальной задержкой* $O(mn)$.

В [18, 18] 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 [18,18] подход к построению алгоритмов дуализации не является новым (алгоритмы RS и MMCS фактически являются асимптотически оптимальными). В [18,18] показано, что алгоритмы RS и MMCS по скорости значительно превосходят ряд других алгоритмов, в частности инкрементальный квазиполиномиальный алгоритм из [4].

В настоящей работе построены асимптотически оптимальные алгоритмы RUNC, RUNC-M, PUNC. Каждый из этих алгоритмов, так же, как и алгоритм ОПТ, основан на перечислении совместимых наборов столбцов, удовлетворяющих дополнительным условиям (для разных алгоритмов эти условия разные). Показано, что предлагаемые алгоритмы требуют меньших временных затрат по сравнению с асимптотически оптимальными алгоритмами, построенными ранее в [7–16] и в большинстве случаев опережают алгоритмы из [18, 18].

Основные принципы работы ранее построенных и новых 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. Примерами служат алгоритмы AO1, AO2, AO2K и AO2M, построенные в [7, 12, 16].

Алгоритмы второго типа основаны на перечислении с полиномиальной задержкой без повторений максимальных совместимых наборов столбцов. Примерами являются алгоритм ОПТ из [14] и алгоритмы MMCS, RS из [18, 18]. Построенные в данной работе алгоритмы дуализации RUNC, RUNC-M и PUNC относятся ко второму типу.

Опишем общую схему работы алгоритма второго 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 $j \in H$, покрывающий i.

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 *запрещенным* 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.

Общая схема алгоритма второго типа. Работу асимптотически оптимальных алгоритмах 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 построенной 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 $X[H]$.

На шаге 1 на итерации 1 по некоторому правилу формируется набор столбцов $C[\varnothing]$ матрицы 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 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 запрещенным 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, непокрытые набором H, то результатом шага становится неприводимое покрытие H', и происходит переход к следующему шагу. В противном случае по некоторому правилу строится набор столбцов $C[H']$, 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, и происходит переход к следующей итерации. В противном случае алгоритм завершает работу. □

Алгоритмы второго типа различаются правилами построения набора $C[\varnothing]$ на шаге 1 на итерации 1 и построения набора $C[H]$ при создании новой вершины H. Опишем эти различия.

Введем дополнительные понятия и 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.

Будем говорить, что строка $i \in R$ является *охватывающей* для строки $t \in R$ в подматрице $L(R, C)$, если $a_{ij} \geqslant a_{tj}$, $\forall j \in C$. Известно, 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.

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 $w_i(C) \geqslant w_t(C)$.

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, непокрытые 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, запрещенные для H'. Затем из $L(R[H'], C[H'])$ последовательно удаляются охватывающие строки и нулевые столбцы, 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. □

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 запрещенным 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, непокрытые 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, непокрытая 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 $C[H']$. Далее 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 предлагаемых алгоритмах RUNC (аббревиатура от Remove Unallowable Columns) и RUNC-M (RUNC Minimum weight row) комбинируются идеи алгоритмов ОПТ и MMCS.

Алгоритм RUNC (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, непокрытые 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, непокрытая 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, запрещенных для H' столбцов. Далее 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. □

Для описания алгоритмов RS и PUNC 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-запрещенным* 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-запрещенным 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-запрещенного столбца j может существовать строка $t > i$ такая, что j не является t-запрещенным.

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, непокрытые 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, непокрытая столбцом j, формируется набор столбцов $C[H']$, не являющихся i-запрещенными для H', и строится подматрица $L(R[H'], C[H'])$, содержащая строки подмат-

рицы $L(R[H], C[H])$, непокрытые столбцом j. Далее 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 основе предлагаемого в настоящей работе 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', $i[H] < i' \leqslant m$, непокрытая 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'-запрещенными для H', с вершиной H' связывается строка $i[H'] = i'$. Далее 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. □

Подробное описание алгоритмов RUNC и RUNC-M

В данном разделе приводится подробное описание алгоритмов RUNC и RUNC-M и анализируется сложность шага этих алгоритмов

Как было отмечено в предыдущем разделе, алгоритмы RUNC и RUNC-M имеют незначительные отличия. Поэтому ниже приводится их общее описание, и даются комментарии относительно конструктивных особенностей и экспериментальных оценок эффективности этих алгоритмов.

Для наглядности описания алгоритмов RUNC и RUNC-M 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, непокрытых набором H, набор столбцов, совместимых с H, и наборы строк, опорных для $(H, j), j \in H$.

Введем глобальные переменные: H — текущий совместимый набор столбцов, $L(R, C)$ — 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 1).

Фактически, алгоритмы RUNC и RUNC-M различаются способом выбора строки i на шаге 2 процедуры `BuildSubtreeRUNC`. Выбор строки i определяет состав дочерних узлов в дереве решений для текущего набора H. Экспериментально установлено, что в большинстве случаев RUNC-M является эффективнее RUNC, поскольку при его использовании существенно сокращается число рекурсивных вызовов процедуры `BuildSubtreeRUNC`

Алгоритм 1 RUNC (RUNC-M)

Вход: L — булева матрица размера $m \times n$;
Выход: $\mathcal{P}(L)$ — набор неприводимых покрытий матрицы L;

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;
$H := \varnothing$;
$R := \{1, \ldots, m\}$;
$D := \{1, \ldots, n\}$;
для всех $l = 1, \ldots, n$
  $S_l := \varnothing$;
вызвать `BuildSubtreeRUNC`; *// построение дерева решений с корнем* $\varnothing$

ПРОЦЕДУРА `BuildSubtreeRUNC`

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

ПРОЦЕДУРА `CreateNodeRUNC`(j)

$S_j := \{i \in R : a_{ij} = 1\}$; *// добавить новые опорные строки*
удалить из R строки, покрытые j;
для всех $(l \in H : S_l \neq \varnothing)$
  удалить из S_l строки, покрытые столбцом j; *// обновить опорные строки*
  если $(\exists i \in S_l : a_{ip} = 0, \forall p \in D)$ **то**
    $S_l := \varnothing$;
  иначе
    для всех $(p \in D)$
      если $(a_{ip} = 1, \forall i \in S_l)$ **то**
        удалить p из D; *// удалить запрещенные столбцы*
добавить j в H;

(число внутренних вершин дерева решений), что компенсирует вычислительные затраты поиска строки с минимальным весом в алгоритме RUNC-M.

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

Алгоритм 2 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, непокрытая 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;*

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: $i := i + 1$;

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, непокрытую набором столбцов D, набор S_l не влияет на состав запрещенных столбцов в D, поэтому на шаге 7 набор S_l заменяется на $\varnothing$. На шагах 9–11 из D удаляются запрещенные для $H \cup \{j\}$ столбцы.

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

Подробное описание алгоритма PUNC

В данном разделе приводится подробное описание алгоритма PUNC и анализируется сложность его шага.

Для наглядности описания алгоритма PUNC 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 обновляет текущее состояние алгоритма: номер строки, непокрытой набором H и набор строк, опорных для $(H, j), j \in H$.

Введем глобальные переменные: H — текущий совместимый набор столбцов, i — текущая строка, непокрытая 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 2).

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 рекурсивно 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, непокрытая 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 [18] и подробное описание построенного алгоритма PUNC, можно заметить следующее. Алгоритм RS после построения очередного набора H формирует (модифицирует) набор строк, непокрытых набором H, и наборы строк $S(H, l), l \in H$. Алгоритм PUNC после построения очередного набора H модифицирует минимальный номер строки i, непокрытой 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 $O(mnq)$, 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.

Результаты экспериментов

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 конечны-

Таблица 1. Случайные матрицы. Случай $m = 10, 50 \leqslant n \leqslant 300$

ОПТ	**0.06**	0.31	**1.2**	**3.2**	8.
MMCS	0.1	0.6	2.3	6.4	15.4
RUNC	**0.06**	**0.3**	1.5	4.3	10.8
RUNC-M	**0.06**	**0.3**	1.4	4.	11.
RS	0.09	0.5	2.2	6.	14.4
PUNC	**0.06**	**0.3**	**1.2**	**3.2**	**7.8**
m	10	10	10	10	10
n	100	150	200	250	300
$\lvert\mathcal{P}(L)\rvert^*$	146248	836107	3257326	8666765	20658771
$\lvert H\rvert^*$	4.1	4.3	4.5	4.6	4.7

Таблица 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
PUNC	0.03	0.16	**0.6**	**2.**	**5.**
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

ми 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 [14]. 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 счета на случайных матрицах приведены в таб. 1–7. Алгоритмы сравнивались в следующих случаях:

1) $n > m$ (см. табл. 1, 2, 3);
2) $n < m$ (см. табл. 4, 5, 6);
3) $n = m$ (см. табл. 7).

Столбцы табл. 1–7 (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 «$|\mathcal{P}(L)|^*$» и «$|H|^*$» 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.

При небольшом количестве строк $m = 10, 20$ среди алгоритмов нет однозначного лидера. В тройку самых быстрых входят алгоритмы ОПТ, PUNC и RUNC-M (см. табл. 1 и 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. 3–7). 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

Таблица 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
PUNC	0.08	0.4	1.6	4.7
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 4. Случайные матрицы. Случай $50 \leqslant m \leqslant 300$, $n = 40$

ОПТ	0.5	1.2	2.2	3.4	5.
MMCS	0.5	1.3	2.4	3.7	5.2
RUNC	0.3	0.9	1.6	2.4	3.5
RUNC-M	**0.24**	**0.6**	**1.**	**1.6**	**2.2**
RS	0.6	1.4	2.5	3.8	5.4
PUNC	0.4	1.2	2.3	3.8	5.8
m	100	150	200	250	300
n	40	40	40	40	40
$\lvert\mathcal{P}(L)\rvert^*$	333177	712507	1155969	1636729	2183946
$\lvert H\rvert^*$	6.3	6.8	7.1	7.4	7.6

Таблица 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
PUNC	0.7	2.4	5.8	10.6	18.1
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 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
PUNC	4.8	8.7	13.8	20.8
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

Таблица 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
PUNC	0.01	0.13	1.5	10.8	70.
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 8. Модельные данные. Граф сочетаний $M(n)$

ОПТ	0.09	0.17	0.3	0.7	1.4	2.9	5.8
MMCS	0.16	0.3	0.7	1.6	3.1	6.3	13.2
RUNC	**0.05**	**0.12**	**0.23**	**0.5**	**1.1**	**2.3**	**4.6**
RUNC-M	0.06	0.14	0.27	0.6	**1.1**	2.4	7.
RS	0.2	0.5	0.9	1.7	4.2	8.6	16.8
PUNC	0.17	0.31	0.6	1.4	2.8	5.8	12.2
m	17	18	19	20	21	22	23
n	35	37	39	41	43	45	47
$\lvert\mathcal{P}(L)\rvert$	131072	262144	524288	1048576	2097152	4194304	8388608
$\lvert H\rvert^*$	17.	18.	19.	20.	21.	22.	23.

Таблица 9. Модельные данные. Двойственный к $M(n)$ гиперграф $DM(n)$

ОПТ	11.2	46.9	184.5	> 600	> 600	> 600	> 600
MMCS	0.9	3.1	10.	35.	122.3	399.9	> 600
RUNC	0.27	0.8	2.5	8.9	28.7	93.6	569.9
RUNC-M	0.28	0.8	2.7	9.5	31.7	109.2	**309.2**
RS	0.6	2.2	6.	20.3	70.	283.6	> 600
PUNC	**0.17**	**0.5**	**1.6**	**5.6**	**17.3**	**51.7**	315.4
m	16384	32768	65536	131072	262144	524288	1048576
n	29	31	33	35	37	39	41
$\lvert\mathcal{P}(L)\rvert$	14	15	16	17	18	19	20
$\lvert H\rvert^*$	2.	2.	2.	2.	2.	2.	2.

RUNC-M. Интересно, что дерево решений алгоритма ОПТ, делающего, как правило, наименьшее число лишних шагов, почти всегда состоит из наибольшего числа вершин. Это обстоятельство наводит на мысль, что число лишних шагов слабо коррелирует со скоростью работы алгоритма.

Тестирование алгоритмов также производилось на модельных данных из [18]. 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 $\mathcal{H}$);
3) $SDFP(n)$ — самодвойственный гиперграф с $n = 7k+2$ вершинами и ребрами вида $\{\{n\} \cup E : E \in (FP(k))^d\} \cup \{\{n-1\} \cup E : E \in FP(k)\} \cup \{\{n-1, n\}\}$, 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 $n/2+1$);
5) $SDTH(n)$ — самодвойственный пороговый гиперграф с n вершинами и $m = \frac{(n-2)^2}{4} + \frac{n}{2} + 1$ ребрами вида $\{\{n\} \cup E : E \in (TH(n-2))^d\} \cup \cup\{\{n-1\} \cup E : E \in TH(n-2)\} \cup \{\{n-1, n\}\}$.

Тестирование алгоритмов также проводилось на прикладных задачах из [18]. Использовались матрицы инцидентности гиперграфов, возникающих в следующих прикладных задачах.

Таблица 10. Модельные данные. Самодвойственный гиперграф $SDPF(n)$

ОПТ	0.05	2.7	200.9	> 600
MMCS	**0.016**	0.5	25.3	> 600
RUNC	**0.016**	0.5	20.4	> 600
RUNC-M	**0.016**	0.3	**9.1**	440.2
RS	**0.016**	0.4	15.	> 600
PUNC	**0.016**	**0.25**	10.4	**329.5**
m	365	2430	16843	117692
n	24	31	38	45
$\|\mathcal{P}(L)\|$	365	2430	16843	117692
$\|H\|^*$	9.6	12.9	16.	19.

Таблица 11. Модельные данные. Пороговый граф $TH(n)$

ОПТ	16.5	42.5	100.2	214.5	412.7
MMCS	0.001	**0.016**	**0.016**	**0.05**	**0.05**
RUNC	0.031	0.05	0.08	0.12	0.19
RUNC-M	0.05	0.05	0.08	0.14	0.2
RS	0.001	**0.016**	**0.016**	**0.05**	0.06
PUNC	0.08	0.14	0.19	0.31	0.5
m	3600	4900	6400	8100	10000
n	121	141	161	181	201
$\|\mathcal{P}(L)\|$	61	71	81	91	101
$\|H\|^*$	89.	104.	119.	134.	149.

Таблица 12. Модельные данные. Самодвойственный гиперграф $SDTH(n)$

ОПТ	42.2	103.2	217.8	411.1	> 600	> 600	> 600	> 600	> 600
MMCS	0.031	**0.05**	0.08	**0.09**	**0.14**	**0.23**	0.4	**0.5**	0.7
RUNC	0.05	0.08	0.11	0.19	0.4	0.7	1.4	2.	3.1
RUNC-M	0.05	0.09	0.12	0.23	0.4	0.9	1.3	2.	3.2
RS	**0.016**	**0.05**	**0.06**	**0.09**	0.16	**0.23**	**0.3**	0.6	**0.6**
PUNC	0.3	0.5	0.7	1.7	2.2	5.7	20.9	12.7	27.6
m	4972	6482	8192	10102	14522	19742	25762	32582	40202
n	143	163	183	203	243	283	323	363	403
$\|\mathcal{P}(L)\|$	4972	6482	8192	10102	14522	19742	25762	32582	40202
$\|H\|^*$	4.5	4.5	4.5	4.5	4.5	4.5	4.5	4.5	4.5

1) На основании данных о сделанных игроками ходах в настольной игре Connect-4 составлены два гиперграфа: WIN и $LOSE$. Данные взяты из UCI Machine Learning Repository [19]. 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.

2) Гиперграфы BMS и ACC сформированы на основе проблемы поиска совместно встречающихся наборов. Использовались прикладные задачи «BMS-WebView-2» и «accidents» из FIMI Repository [20]. 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 счета на модельных данных приведены в табл. 8–12. Результаты счета на прикладных задачах приведены в табл. 13–16. В названиях табл. 8–16 указаны типы задач. Столбцы табл. 8–16 (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

Таблица 13. Прикладные задачи. Гиперграф $WIN(m)$

ОПТ	1.5	27.8	517.7	> 600	> 600	> 600	> 600
MMCS	0.06	0.3	2.9	12.	50.5	212.8	> 600
RUNC	0.05	0.23	2.3	13.	57.9	511.	> 600
RUNC-M	**0.031**	**0.17**	**1.3**	**5.1**	**20.3**	**75.5**	**306.5**
RS	0.08	0.5	5.3	21.4	84.7	> 600	> 600
PUNC	0.22	1.7	20.2	104.	530.9	> 600	> 600
m	800	1600	3200	6400	12800	25600	44473
n	78	80	82	82	84	84	85
$\lvert\mathcal{P}(L)\rvert$	11675	71840	459502	1277933	4587967	11614885	31111249
$\lvert H\rvert^*$	15.1	16.7	17.9	18.9	19.9	20.8	21.9

Таблица 14. Прикладные задачи. Гиперграф $LOSE(m)$

ОПТ	1.3	4.2	66.5	> 600	> 600	> 600	> 600
MMCS	0.11	0.3	1.1	12.9	41.8	185.8	519.5
RUNC	0.09	0.5	2.2	10.3	58.5	278.6	> 600
RUNC-M	**0.05**	**0.17**	**0.5**	**4.3**	**12.5**	**49.2**	**137.7**
RS	0.16	0.5	1.7	17.1	107.1	449.7	> 600
PUNC	0.4	1.5	5.8	71.1	501.2	> 600	> 600
m	400	800	1600	3200	6400	12800	16635
n	79	81	81	81	81	85	85
$\lvert\mathcal{P}(L)\rvert$	33087	79632	212761	2396735	4707877	16405082	39180611
$\lvert H\rvert^*$	13.7	14.8	15.9	17.2	17.6	19.3	20.1

Таблица 15. Прикладные задачи. Гиперграф $ACC(m)$

ОПТ	0.05	0.23	1.1	8.2	136.3
MMCS	0.06	0.27	0.8	3.7	33.
RUNC	0.031	0.09	**0.25**	**0.9**	7.6
RUNC-M	**0.016**	**0.08**	**0.25**	**0.9**	**7.3**
RS	0.05	0.3	0.8	3.4	35.3
PUNC	0.031	0.27	1.4	9.9	208.3
m	2000	4322	10968	32207	135439
n	81	336	336	336	442
$\lvert\mathcal{P}(L)\rvert$	3547	7617	17486	47137	185218
$\lvert H\rvert^*$	4.7	5.1	5.7	6.5	7.3

Таблица 16. Прикладные задачи. Гиперграф $BMS(m)$

ОПТ	**0.7**	3.9	**21.1**	105.3	310.6
MMCS	1.3	15.7	117.4	510.8	> 600
RUNC	0.8	**3.8**	24.7	**94.2**	**261.7**
RUNC-M	0.8	4.	25.1	113.6	315.6
RS	1.3	14.2	98.2	446.7	> 600
PUNC	3.5	34.8	217.4	> 600	> 600
m	823	2591	6946	17315	30405
n	3340	3340	3340	3340	3340
$\lvert\mathcal{P}(L)\rvert$	89448	438867	1289303	2297560	3064937
$\lvert H\rvert^*$	2.	2.	2.	2.	2.1

содержат 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 счета 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 ОПТ и RUNC-M. По всей видимости, скорость работы достигается за счет «удаления запрещенных 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 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.

С задачей дуализации самодвойственого графа $SDFP(n)$ снова лучше всех справился алгоритм PUNC, от которого в основном незначительно отстает алгоритм RUNC-M.

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

Тестирование на прикладных задачах показывает, что лучшим является алгоритм RUNC-M, преимущество которого особенно очевидно на входных матрицах большого размера.

Заключение

В рамках подхода, разработанного Дюковой Е.В. и развитого ей и ее учениками, проведен анализ асимптотически оптимальных алгоритмов дуализации, построенных в отечественных [7–16] и зарубежных [18, 18] публикациях. Выделено два типа алгоритмов:

1) перечисляющие с полиномиальной задержкой с повторениями наборы столбцов, порождаемые максимальными единичными подматрицами;
2) перечисляющие с полиномиальной задержкой без повторений максимальные совместимые наборы столбцов.

Дана общая схема асимптотически оптимальных алгоритмов дуализации второго типа. В терминах предложенной схемы описаны построенные ранее алгоритмы ОПТ из [14], MMCS и RS из [18,18] и новые алгоритмы RUNC, RUNC-M и PUNC. Приведены подробные описания новых алгоритмов с использованием рекурсивных процедур, и проанализирована временная сложность их шага.

Проведенные эксперименты показывают, что наблюдается определенная специализация по типам задач, и абсолютного лидера среди тестируемых алгоритмов нет. Однако в большинстве случаев самым быстрым является алгоритм RUNC-M. Счет, проведенный в настоящей работе и в [18,18], демонстрирует приемущество асимптотически оптимальных алгоритмов дуализации по сравнению с алгоритмами, имеющими другую конструкцию, в частности с инкрементальными квазиполиномиальными алгоритмами.

Литература

[1] *Khachiyan L., Boros E., Elbassioni K., Gurvich V.* An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals // Discrete Applied Mathematics, 2006. Vol. 154. № 16. P. 2350–2372. URL: `http://linkinghub.elsevier.com/retrieve/pii/S0166218X06001910`.

[2] *Eiter T., Gottlob G., Makino K.* New results on monotone dualization and generating hypergraph transversals // SIAM J. Computing, 2003. Vol. 32. № 2. P. 514–537.

[3] *Fredman M.L., Khachiyan L.* On the complexity of dualization of monotone disjunctive normal forms // J. Algorithms, 1996. Vol. 21. P. 618–628.

[4] Frequent Itemset Mining Dataset Repository. http://fimi.ua.ac.be/data/.

[5] *Boros E., Elbassioni K.* Generating maximal independent sets for hypergraphs with bounded edge-intersections // LATIN 2004: Theoretical Informatics. — Springer Berlin Heidelberg, 2004. P. 488–498. URL: http://www.springerlink.com/index/b9t9wx6ue9t9hvw3.pdf.

[6] *Boros E., Gurvich V., Elbassioni K., Khachiyan L.* An efficient incremental algorithm for generating all maximal independent sets in hypergraphs of bounded dimension // Parallel Processing Letters, 2000. Vol. 10. No. 4. P. 253–266. URL: BibUrlhttp://www.worldscientific.com/doi/abs/10.1142/S0129626400000251.

[7] *Дюкова Е.В.* Об асимптотически оптимальном алгоритме построения тупиковых тестов // ДАН СССР, 1977. Т. 233. №4. С. 527–530.

[8] *Djukova E.V.* Discrete recognition procedures: The complexity of realization // Pattern Recognition and Image Analysis, 2003. Vol. 13. No. 1. P. 8–10.

[9] *Djukova E.V., Sotnezov R.M.* On the complexity of discrete generation problems // Doklady Mathematics, 2010. Vol. 82. No. 3. P. 847–849.

[10] *Djukova E.V., Zhuravlev Yu.I.* Discrete methods of information analysis in recognition and algorithm synthesis // Pattern Recognition and Image Analisys, 1997. Vol. 7. № 2. P. 192–207.

[11] *Дюкова Е.В.* О сложности реализации некоторых процедур распознавания // Ж. вычисл. матем. и матем. физ. 1987. Т. 27. №1. С. 114–127.

[12] *Дюкова Е.В.* О сложности реализации дискретных (логических) процедур распознавания // Ж. вычисл. матем. и матем. физ., 2004. Т. 44. №3. С. 562–572.

[13] *Дюкова Е.В.,Журавлев Ю.И.* Дискретный анализ признаковых описаний в задачах распознавания большой размерности // Ж. вычисл. матем. и матем. физ., 2000. Т. 40. №8. С. 1264–1278.

[14] *Дюкова Е.В., Инякин А.С.* Асимптотически оптимальное построение тупиковых покрытий целочисленной матрицы // Математические вопросы кибернетики, 2008. Т. 17. С. 235–246.

[15] *Дюкова Е.В., Сотнезов Р.М.* О сложности задачи дуализации // Ж. вычисл. матем. и матем. физ., 2012. Т. 52. №10. С. 1926–1935.

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

[17] *Murakami K., Uno T.* Efficient algorithms for dualizing large-scale hypergraphs // 15th Meeting on Algorithm Engineering and Experiments, ALENEX 2013 Proceedings. New Orleans, Louisiana, USA, 2013. *SIAM*, 2013.

[18] *Murakami K., Uno T.* Efficient algorithms for dualizing large-scale hypergraphs // Discrete Applied Mathematics, 2014. Vol. 170. P. 83–94.

[19] UCI machine learning repository. http://archive.ics.uci.edu/ml/

[20] *Johnson D., Yannakakis M., Papadimitriou C.* On generating all maximal independent sets // Information Processing Letters, 1988. Vol. 27. № 3. P. 119–123. URL: http://www.sciencedirect.com/science/article/pii/0020019088900658.

References

[1] *Johnson D., Yannakakis M., Papadimitriou C.* 1988. On generating all maximal independent sets. *Information Processing Letters* 27(3):119–123. URL: `http://www.sciencedirect.com/science/article/pii/0020019088900658`.

[2] *Eiter T., Gottlob G., Makino K.* 2003. New results on monotone dualization and generating hypergraph transversals. *SIAM J. Computing* 32(2):514–537.

[3] *Fredman M. L., Khachiyan L.* 1996. On the complexity of dualization of monotone disjunctive normal forms. *J. Algorithms* 21:618–628.

[4] *Khachiyan L., Boros E., Elbassioni K., Gurvich V.* 2006. An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals and its application in joint generation. *Discrete Applied Mathematics* 154(16):2350–2372. URL: `http://linkinghub.elsevier.com/retrieve/pii/S0166218X06001910`.

[5] *Boros E., Elbassioni K.* 2004. Generating maximal independent sets for hypergraphs with bounded edge-intersections. *LATIN 2004: Theoretical Informatics*. Berlin–Heidelberg: Springer. 488–498. URL: `http://www.springerlink.com/index/b9t9wx6ue9t9hvw3.pdf`.

[6] *Boros E., Gurvich V., Elbassioni K., Khachiyan L.* 2000. An efficient incremental algorithm for generating all maximal independent sets in hypergraphs of bounded dimension. *Parallel Processing Letters* 10(4):253–266. URL: BibUrlhttp://www.worldscientific.com/doi/abs/10.1142/S0129626400000251.

[7] *Djukova E.V.* 1977. Asymptotically optimal algorithm for constricting irredundant tests. *Dokl. Akad. Nauk SSSR* 233(4):527–530.

[8] *Djukova E.V.* 2003. Discrete recognition procedures: The complexity of realization. *J. Pattern Recognition and Image Analysis* 13(1):8–10.

[9] *Djukova E.V., Sotnezov R.M.* 2010. On the complexity of discrete generation problems. *Doklady Mathematics* 82(3):847–849.

[10] *Djukova E.V., Zhuravlev Yu.I.* 1997. Discrete methods of information analysis in recognition and algorithm synthesis. *Pattern Recognition and Image Analisys* 7(2):192–207.

[11] *Djukova E.V.* 1987. The complexity of the realization of certain recognition procedures. *USSR Comput. Math. Math. Phys.* 27(1):74–83.

[12] *Djukova E.V.* 2004. On the implementation complexity of discrete (logical) recognition procedures. *Comput. Math. Math. Phys.* 44(3):532–541.

[13] *Djukova E.V., Zhuravlev Yu.I.* 2000. Discrete analysis of feature descriptions in recognition problems of high dimensionality. *Comput. Math. Math. Phys.* 40(8):1214–1227.

[14] *Djukova E.V., Inyakin A.S.* 2008. Asymptotically optimal constriction of irredundant coverings of an integer matrix. *Mathematical problems in cybernetics*. Moscow: Nauka. 17:235–246. [In Russian.]

[15] *Djukova E.V., Sotnezov R.M.* 2012. On the complexity of the dualization problem. *Comput. Math. Math. Phys.* 52(10):1472–1481.

[16] *Djukova E.V., Prokofjev P.A.* 2014. On asymptotically optimal enumeration for irreducible coverings of Boolean matrix. *Applied Discrete Mathematics* 1:96–105. [In Russian.]

[17] *Keisuke Murakami, Takeaki Uno.* 2011. Efficient algorithms for dualizing large-scale hypergraphs. arXiv: CoRR abs/1102.3813.

[18] *Keisuke Murakami, Takeaki Uno.* 2013. Efficient algorithms for dualizing large-scale hypergraphs. *15th Meeting on Algorithm Engineering and Experiments, ALENEX 2013 Proceedings*. New Orleans, Louisiana, USA. *SIAM* 2013.

[19] UCI machine learning repository. http://archive.ics.uci.edu/ml/.

[20] Frequent Itemset Mining Dataset Repository. http://?mi.cs.helsinki.?/data/.