Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра математических методов прогнозирования

НИКИФОРОВ Андрей Геннадьевич

Разработка подхода к построению эффективных параллельных алгоритмов для дискретных перечислительных задач

ДИПЛОМНАЯ РАБОТА

Научный руководитель:
д.ф-м.н., доцент кафедры ММП
Е.В. Дюкова

Москва, 2015

Оглавление

1. Введение

Одной из фундаментальных задач дискретной математики является дуализация, которая в матричной формулировке звучит следующим образом.

Дана булева матрица L размера $m \times n$. Набор H, состоящий из различных столбцов матрицы L, называется неприводимым покрытием, если он удовлетворяет двум условиям: (1) в подматрице L^H матрицы L, образованной столбцами набора H, не содержится строки, состоящей из одних нулей; (2) подматрица L^H содержит каждую из строк вида $(1,0,0,\dots,0)$, $(0,1,0,\dots,0)$, … $(0,0,0,\dots,1)$, т.е. с точностью до перестановки строк содержит единичную подматрицу порядка $|H|$. Если набор H удовлетворяет условию (1), то он называется покрытием. Если набор H удовлетворяет условию (2), то он называется совместимым. Требуется построить множество $P(L)$ всех неприводимых покрытий матрицы L.

Дуализация имеет и другие эквивалентные формулировки [1]. Приведём основные из них.

1. Дана конъюнктивная нормальная форма, реализующая монотонную булеву функцию F. Требуется построить сокращенную дизъюнктивную нормальную форму функции F.
2. Дан гиперграф G. Требуется перечислить все минимальные трансверсали гиперграфа G.

Двойственной к задаче 2 является задача перечисления всех минимальных вершинных покрытий гиперграфа.

Дуализация возникает во многих областях дискретной математики (комбинаторике, теории гиперграфов, целочисленном программировании), в теории игр, в теории баз данных, в машинном обучении и т.д.

Асимптотические оценки типичных значений числа решений дуализации [2] показывают, что, как правило, это число растёт экспоненциально с ростом размера

входных данных. Поэтому задача дуализации относится к числу труднорешаемых. Сложность алгоритмов для труднорешаемых перечислительных задач принято оценивать сложностью шага. Существует несколько подходов к оценке эффективности алгоритмов для таких задач.

Говорят, что алгоритм работает с (квази) полиномиальной задержкой, если на каждом шаге строится ровно одно решение и сложность этого шага ограничивается (квази) полиномом от размера входных данных. В матричной формулировке дуализации под входом понимается матрица L, а под размером входных данных – числа m и n. Алгоритмы дуализации с (квази) полиномиальной задержкой удалось построить только для некоторых частных случаев [3] и [4] (например, когда в каждой строке исходной матрицы не более двух единиц).

Говорят, что алгоритм является инкрементальным (квази) полиномиальным, если на каждом шаге строится ровно одно решение и сложность очередного шага ограничена (квази) полиномом от размера входных данных и числа уже найденных решений. Наилучший результат для дуализации, касающийся построения инкрементальных алгоритмов, получен Л.Г. Хачияном с соавторами в работах [5] и [6], в которых построен инкрементальный алгоритм с квазиполиномиальной временной оценкой $O(N^{\log(N)})$.

В [2] и [7] Е.В. Дюковой предложен подход к построению асимптотически оптимальных алгоритмов для перечислительных задач, проиллюстрированный на примере дуализации в матричной формулировке. В дальнейшем этот подход получил развитие в [8], [9] и [10]. Асимптотически оптимальные алгоритмы дуализации с полиномиальной задержкой являются эффективными «в типичном случае». На каждом шаге этих алгоритмов строится набор столбцов матрицы, удовлетворяющий условию совместимости (2). В отличие от алгоритмов дуализации с (квази) полиномиальной задержкой, указанные алгоритмы могут делать «лишние» шаги, причём доля таких шагов стремится к нулю для почти всех матриц L, размера $m \times n$, при $m, n \to \infty$. На «лишнем» шаге либо 1) строится

набор столбцов матрицы, не являющийся покрытием, либо 2) строится набор столбцов, найденный ранее. Заметим, что при данном подходе не требуется просматривать уже найденные решения. Проверка на повторяемость осуществляется за полиномиальное время от m и n. Асимптотически оптимальные алгоритмы дуализации можно разделить на три типа: алгоритмы первого типа делают «лишние» шаги вида 1), алгоритмы второго типа – шаги вида 2), алгоритмы третьего типа – шаги вида 1) и 2).

В отличие от инкрементальных алгоритмов, асимптотически оптимальные алгоритмы дуализации имеют практическое применение. В [11] и [12] показано, что наиболее быстро среди этих алгоритмов работают алгоритмы, не делающие «повторных» шагов (алгоритмы первого типа). Из рассмотренных в [11] и [12] алгоритмов лидерами по скорости счёта являются PUNC [12] и RUNC-M [12]. Там же показано, что прежний лидер, алгоритм ОПТ [13], незначительно уступает в некоторых случаях PUNC и RUNC-M.

В силу вычислительной сложности дуализации, актуальным является использование параллельных вычислений. Существуют простые и очевидные схемы распараллеливания асимптотически оптимальных алгоритмов дуализации, основным недостатком которых является неравномерная загрузка процессоров, что приводит и к недостаточному ускорению времени работы параллельного алгоритма по сравнению с его последовательной версией. Схема распараллеливания определяется способом выбора вычислительных подзадач и способом распределения этих подзадач между процессорами.

Целью данной работы является разработка подхода к построению эффективных параллельных асимптотически оптимальных алгоритмов дуализации.

В рамках исследования **решены следующие задачи**:

1. За счет более существенного использования побитовых операций эффективно реализована последовательная версия асимптотически оптимального алгоритма дуализации ОПТ.

2. На базе алгоритма ОПТ разработан асимптотически оптимальный алгоритм дуализации ОПТ1, который работает значительно быстрее других известных алгоритмов дуализации.
3. Для параллельных алгоритмов дуализации предложены способы формирования подзадач и распределения их между вычислительными узлами, основанные на экспериментальных статистических оценках объёмов этих подзадач. Разработаны две схемы распараллеливания алгоритма ОПТ.
4. Реализованы две параллельные версии алгоритма ОПТ, и проведено их тестирование на суперкомпьютере факультета ВМК МГУ имени М. В. Ломоносова IBM Blue Gene/P.

При реализации последовательной версии алгоритма ОПТ на языке C++ применены конструктивные приёмы, позволившие сделать эту реализацию эффективной, а именно побитовые представление данных и динамический выбор определённых функций. Предлагаемая реализация особенно эффективна при $m \ll n$ (например, при $m = 30$ и $n = 150$) и превосходит реализации алгоритмов ОПТ, PUNC и RUNC-M из [12] на 20-40%.

Разработан асимптотически оптимальный алгоритм ОПТ1, являющийся модификацией алгоритма ОПТ. Новый алгоритм делает значительно меньше «лишних» шагов по сравнению с алгоритмом ОПТ, и значительно превосходит по скорости счёта все рассмотренные реализации других алгоритмов, а именно новой реализации алгоритма ОПТ и реализации ОПТ, PUNC и RUNC-M из [12].

Основным результатом данной работы является разработка оригинальных схем распараллеливания асимптотически оптимальных алгоритмов дуализации. Следует отметить, что за рубежом при создании параллельных алгоритмов дуализации первостепенное внимание уделяется теоретическим оценкам их сложности в зависимости от числа используемых процессоров, [6], [14], [15]. Причём, как правило, строятся алгоритмы, ориентированные на частные случаи, например, когда число единиц в каждой

строке исходной матрицы ограничено некоторой небольшой константой. Кроме того, не приводятся данные о тестировании созданных алгоритмов.

В настоящей работе разработаны две схемы распараллеливания асимптотически оптимальных алгоритмов дуализации (B-схема [16] и S-схема), которые нацелены на общий случай. Их работа проиллюстрирована на примере алгоритма ОПТ1. Особое внимание уделяется практической применимости разработанных параллельных алгоритмов, исследуются балансировка нагрузки между процессорами и ускорение времени работы параллельного алгоритма при увеличении числа процессоров.

Далее рассматривается матричная формулировка задачи дуализации.

Пусть H – неприводимое покрытие матрицы L, состоящее из столбцов $\{j_1, \dots, j_r\}$ и $j_1 < \dots < j_r$. Тогда H назовём j_1 -неприводимым покрытием. Через $P_j(L),\ j \in \{1, \dots, n\}$, обозначим множество j -неприводимых покрытий матрицы L. В разработанных схемах распараллеливания объем подзадач определяется величинами $\nu_j(L) = \left|P_j(L)\right|/|P(L)|$.

Обе схемы имеют статический характер: распределение подзадач происходит заранее и осуществляется по расписанию. С этой целью решается специальная задача оптимизации. Высокое ускорение достигается за счёт использования статистических оценок для объемов подзадач и минимизации накладных расходов на межпроцессорное взаимодействие, что выгодно отличает данный подход от предыдущих отечественных разработок, в которых для достижения равномерности распараллеливание происходило не на первом ярусе дерева решений. Это приводило к достаточно большим накладным расходам и не достигалось максимальное ускорение.

В B-схеме на пространстве элементарных событий $\Omega_1 = \{(L, H) | L \in M_{mn},\ H \in P(L)\}$ с равновероятными исходами вводится случайная величина $\eta(L, H)$, равная j, если $H \in P_j(L),\ j \in \{1, \dots, n\}$. При помощи критерия Хи-квадрат проверяется гипотеза о виде распределения $H_0: f(j) = \psi_{\alpha\beta}(j)$, где $f(j)$ – вероятность события $\eta(L, H) = j$, а $\psi_{\alpha\beta}(j)$ – функция вероятности бета-биномиального распределения с параметрами α и β, которые

вычисляются при помощи метода максимального правдоподобия по случайной выборке из Ω_1. B-схема использует $\psi_{\alpha\beta}(j)$ в качестве приближения искомой величины $\nu_j(L)$.

Опишем S-схему. Пусть L – матрица размера $m \times n$, $r \in \{1, \dots, m\}$ и W_m^r – множество всех подмножеств множества $\{1, \dots, m\}$ мощности r. Через L^w, $w \in W_m^r$, обозначим подматрицу матрицы L, составленную из строк матрицы L с номерами из w. Так же, как в B-схеме, вводится пространство элементарных событий $\Omega_2^r = \{(L^w, H) | w \in W_m^r,\ H \in P(L^w)\}$ с равновероятными исходами и случайная величина $\eta_r(L^w, H)$, равная j, $j \in \{1, \dots, n\}$, если $H \in P_j(L^w)$. При помощи критерия Хи-квадрат проверяется гипотеза о виде распределения $H_0: f_r(j) = \nu_j(L)$, где $f_r(j)$ – вероятность события $\eta_r(L^w, H) = j$. Эмпирические оценки для $f_r(j)$ используются в S-схеме как приближение величины $\nu_j(L)$.

При тестировании параллельных алгоритмов в работе исследуется их сильная масштабируемость (зависимость основных показателей работы параллельного алгоритма от числа процессоров при фиксированном размере задачи). Обе схемы демонстрируют ускорение, близкое к максимальному, и достаточно равномерную загрузку процессоров. B-схема требует значительно больших временных затрат, чем S-схема, связанных с получением оценок для объемов подзадач. Обе схемы перестают быть эффективными при большом числе процессоров, так как распараллеливание происходит на первом ярусе решающего дерева, которое строит асимптотически оптимальный алгоритма ОПТ, и объёмы подзадач сильно не равномерны.

2. Описание асимптотически оптимального алгоритма дуализации ОПТ

В данном разделе приводится описание асимптотически оптимального алгоритма дуализации ОПТ. Подробно описана структура дерева решений, которое строит данный алгоритм, и приведён пример конкретного дерева решений.

Обозначим через M_{mn} множество булевых матриц размера $m \times n$, а через J_u множество $\{1,2,\dots,u\}$. Пусть $L = (a_{ij}) \in M_{mn}$. В данном разделе будем отождествлять набор столбцов (строк) матрицы L с набором их номеров.

Будем говорить, что столбец j (столбец с номером j) покрывает строку i (строку с номером i) матрицы L, если $a_{ij} = 1$.

Строка i матрицы L является опорной для пары $(H, j), j \in H$, если $a_{ij} = 1$ и $a_{iu} = 0$ при $u \in H \setminus \{j\}$. Очевидно, набор H является совместимым тогда и только тогда, когда для каждой пары (H, j), $j \in H$, существует опорная строка. Множество всех опорных строк для (H, j) обозначим через $S(H, j)$.

Столбец j матрицы L называется запрещённым для совместимого набора H, если существует $u \in H$ такой, что столбец j покрывает все строки из $S(H, u)$. В противном случае, будем говорить, что столбец j совместим с набором H. Очевидно, набор $H \cup \{j\}$ совместим тогда и только тогда, когда столбец j совместим с набором H.

Будем говорить, что строка i_1 охватывает строку i_2 в матрице L для набора столбцов C, если для каждого j из этого набора $a_{i_1 j} \geq a_{i_2 j}$.

Асимптотически оптимальный алгоритм дуализации ОПТ перечисляет с полиномиальной задержкой $O(qm^2 n), q = \min(m, n)$, некоторое подмножество совместимых наборов столбцов матрицы L, содержащее множество $P(L)$. Алгоритм строит дерево решений, совершая его обход в глубину. Построение одной висячей вершины – это шаг алгоритма.

Вершина (H, R, C) дерева решений описывается совместимым набором столбцов H, набором строк R и набором столбцов C. В висячей вершине имеет место один из двух случаев: 1) $R = \emptyset$; 2) $R \neq \emptyset$, $C = \emptyset$. В первом случае H – неприводимое покрытие. Во втором случае висячая вершина соответствует «лишнему» шагу. Корню дерева соответствует $(H = \emptyset,\ R = J_m,\ C = J_n)$. Пусть построена внутренняя (не висячая)

вершина (H, R, C), тогда переход к следующей построенной вершине будет осуществляться путём добавления к H столбца из C и удаления некоторых строк и столбцов из R и C, соответственно.

Опишем рекурсивную процедуру *BuildSubtreeOPT*(L, H_0, R_0, C_0) (см. процедура 1) построения поддерева решений. Для запуска алгоритма эту функцию следует вызывать с параметрами $H = \emptyset,\ R = J_m,\ C = J_n$. Отметим, что все аргументы процедуры передаются по значению, или копируются, в связи с чем рекурсивный вызов *BuildSubtreeOPT*(L, H, R, C) никак не изменит текущее значение переменных.

Процедура 1. *BuildSubtreeOPT*(L, H_0, R_0, C_0)

---

1. FOR $j \in \mathrm{C}_0$
2. $R = R_0$
3. $C_0 = C_0 \setminus \{j\}$
4. $C = C_0$
5. $H = H_0 \cup \{j\}$
6. Удалить из R покрытые столбцом j строки
7. IF $R = \emptyset$
8. Сохранить набор H, который является неприводимым покрытием
9. ELSE
10. Удалить из C запрещённые столбцы для набора H
11. Удалить из R охватывающие строки для набора C
12. Удалить из C нулевые столбцы для набора R: $\mathrm{C} = \mathrm{C} \setminus \{j \in C | \forall i \in R: a_{ij} = 0\}$
13. *BuildSubtreeOPT*(L, H, R, C)
14. ENDIF
15. ENDFOR

На рисунке 1 приведён пример дерева решений, построенного алгоритмом ОПТ, для матрицы

$$L = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

3. Новый асимптотически оптимальный алгоритм дуализации ОПТ1

В данном разделе приводится описание асимптотически оптимального алгоритма дуализации ОПТ1. Этот алгоритм является модификацией алгоритма ОПТ и работает значительно быстрее не только алгоритма ОПТ, но и других алгоритмов: PUNC и RUNC-M (см. раздел 4).

Опишем рекурсивную процедуру *BuildSubtreeOPT1*(L, H_0, R_0, C_0) (см. процедура 2) построения поддерева решений для алгоритма ОПТ1. Для запуска алгоритма эту функцию следует вызывать с параметрами $H = \emptyset,\ R = J_m,\ C = J_n$. Отметим, что все аргументы процедуры передаются по значению, или копируются, в связи с чем рекурсивный вызов *BuildSubtreeOPT1*(L, H, R, C) никак не изменит текущее значение переменных.

Пусть построена внутрення вершина (H_0, R_0, C_0). Тогда алгоритм ОПТ при построении следующей вершины добавит к H_0 первый по порядку столбец из C_0. Алгоритм ОПТ1 добавит первый по порядку столбец из множества $C_0^i = \{j \in C_0 | a_{ij} = 1\}$, где i – некоторый номер строки из R_0.

В алгоритме ОПТ1 номер строки $i \in R_0$ выбирается из соображений минимизации числа поддеревьев данной вершины дерева решений. Другими словами, выбирается такой номер $i_{min} \in R_0$ с наименьшей суммой $\sum_{j \in C_0} a_{ij}$. Множество $C_0^{i_{min}}$ в таком случае обозначается через C_0^{min} и является минимальным по мощности среди $C_0^i,\ i \in R_0$.

Для того, чтобы схемы распараллеливания, описанные в разделах 5 и 6, были применимы к алгоритму ОПТ1, требуется его немного модифицировать: на первом ярусе

дерева решений вместо множества C_0^{min} берётся множество $C_0 = \{1,2,\dots,n\}$. В разделе 4 приведены результаты тестирования именно такой последовательной версии ОПТ1.

Процедура 2. *BuildSubtreeOPT1*(L, H_0, R_0, C_0)

1. $C_0^{min} = \{j \in C_0 | a_{i_{min}j} = 1\}$, где $i_{min} = \arg\min_{i \in R_0} \sum_{j \in C_0} a_{ij}$
2. FOR $j \in C_0^{min}$
3. $R = R_0$
4. $C_0 = C_0 \setminus \{j\}$
5. $C = C_0$
6. $H = H_0 \cup \{j\}$
7. Удалить из R покрытые столбцом j строки
8. IF $R = \emptyset$
9. Сохранить набор H, который является неприводимым покрытием
10. ELSE
11. Удалить из C запрещённые столбцы для набора H
12. Удалить из R охватывающие строки для набора C
13. Удалить из C нулевые столбцы для набора R: $\mathrm{C} = \mathrm{C} \setminus \{j \in C | \forall i \in R: a_{ij} = 0\}$
14. *BuildSubtreeOPT2*(L, H, R, C)
15. ENDIF
16. ENDFOR

На рисунке 2 приведён пример дерева решений, построенного алгоритмом ОПТ1, для такой же, как в примере раздела 2, матрицы

$$L = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

У внутренних вершин жирном шрифтом отмечен $i_{min} \in R$.

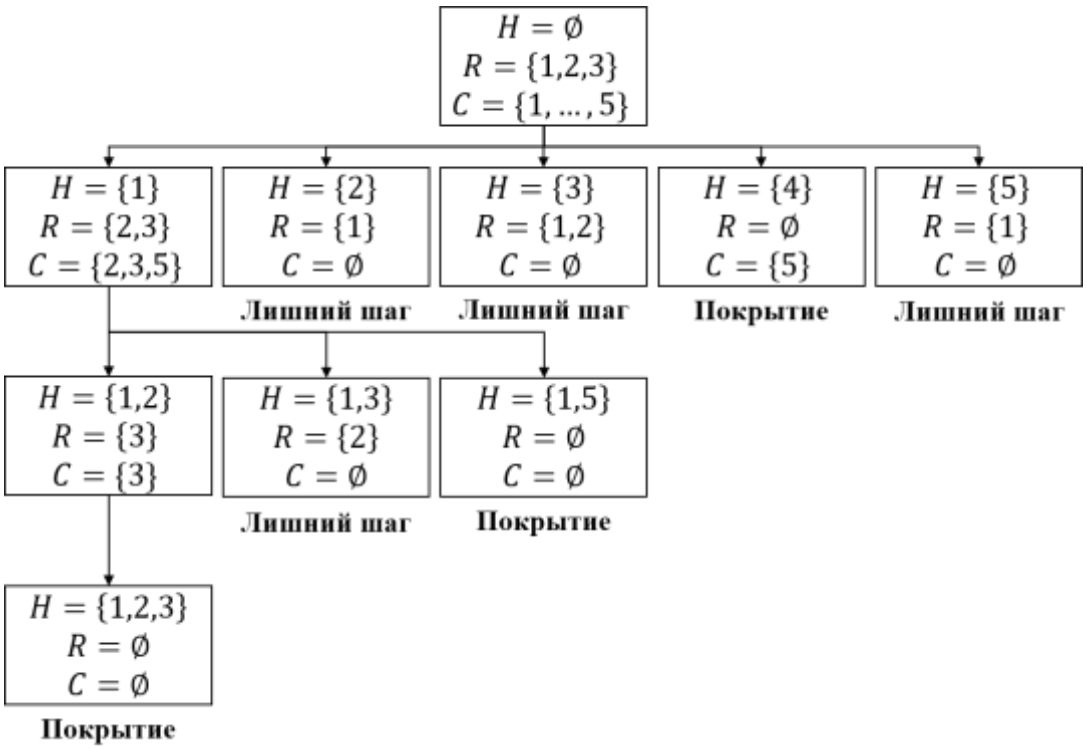


Рис. 1. Пример дерева решений алгоритма ОПТ

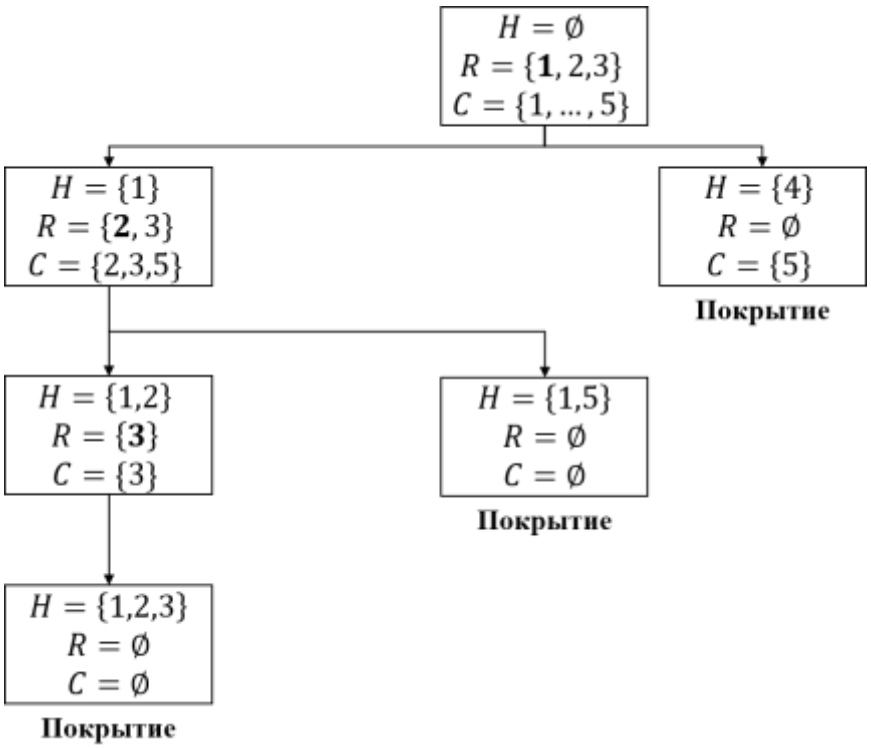


Рис. 2. Пример дерева решений алгоритма ОПТ1

Из рассмотрения рисунка 2 видно, что алгоритм ОПТ1 не делает ни одного «лишнего» шага (отметим, что в общем случае «лишние» шаги останутся). Если сравнивать рисунки

1 и 2, то легко заметить, что дерево решений, которое строит ОПТ1, является «разреженной» версией дерева, которое строит ОПТ.

4. Реализация последовательных версий алгоритмов ОПТ и ОПТ1

В данном разделе перечислены некоторые ключевые особенности предлагаемых реализаций последовательных алгоритмов ОПТ и ОПТ1, дано описание среды тестирования и результатов сравнения предложенных реализаций с другой его реализацией ОПТ [12], а также с реализациям алгоритмов PUNC [12], RUNC-M [12], которые доступны по адресу sourceforge.net/p/logicalanalyze/code/HEAD/tree/trunk/dualization/.

4.1. Особенности реализаций

Настоящие реализации написаны на языке C++. Главной их особенностью является то, что почти все данные хранятся в виде бинарных векторов. Например, элементы каждой строки исходной матрицы L представляются в виде массива длиной $\lceil n/32 \rceil$ двойных машинных слов. Кроме того, программы хранят транспонированную матрицу L^T, представленную аналогично. Такой подход позволяет компактно хранить данные (они целиком умещаются в кэш-памяти процессора) и обеспечить параллельную обработку большого числа элементов матрицу. К примеру, проверка, охватывает ли строка i_1 строку i_2, выполняется за $4\lceil n/32 \rceil + 3$ следующих операций: побитовое «и», побитовое «или», побитовое дополнение, сравнение с нулём.

Многие операции, совершаемые в ОПТ и ОПТ1, допускают несколько вариантов реализации. Например, для удаления запрещённых строк используется две различных функции: первая обрабатывает матрицу по строкам, вторая – по столбцам. Выбор из двух функций осуществляется из соображений минимизации числа операций (в некоторых случаях, такой выбор ускоряет работу алгоритма на 20-30%, несмотря на накладные расходы).

4.2. Среда тестирования

Тестирование последовательных реализаций алгоритмов ОПТ, ОПТ1, PUNC и RUNC-M проводилось на компьютере со следующими характеристиками: Intel Core i5-4200H (2.8 GHz), RAM 12 GB (1600 MHz), Windows 8.1. Для компиляции использовался компилятор Microsoft Visual C++ 2013 Professional (режим полной оптимизации всей программы с использованием интринсиков, флаги /Ox /GL /Oi /Ot). При запуске алгоритмов вывод происходил в нулевой файл (NUL) для того, чтобы не учитывать время взаимодействия с жестким диском (отметим, что при этом функции вывода вызывались).

4.3. Тестирование реализации ОПТ

В таблице 1 приведено время работы реализаций алгоритмов ОПТ, ОПТ1, PUNC и RUNC-M при различных m и n. Эксперименты проводились на случайных булевых матрицах с равновероятным появлением нулей и единиц. Для каждой пары (m,n) обсчитывалось по 20 случайных матриц (по 10 для матриц $30 \times 200,\ 40 \times 200,\ 120 \times 120$) и вычислялись среднее число неприводимых покрытий и среднее время работы алгоритмов (под средним понимается медиана).

Для предложенной реализации ОПТ можно сделать следующие выводы.

1. При $m < n$ достигается наибольший прирост производительности (на 40%) по сравнению с предыдущей реализацией ОПТ, например, для матриц размера 30×150 и 40×150. Кроме того, новая реализация ОПТ работает быстрее реализации алгоритма PUNC [12] на 10-30%.
2. При $m = n$ время работы новой реализации ОПТ практически совпадает со временем работы алгоритма RUNC-M.
3. При $m > n$ новая реализация ОПТ уступает старой реализации на 10-20%
4. Не все приёмы, применявшиеся в новой реализации ОПТ, использовались при программировании алгоритмов ОПТ [12], PUNC [12] и RUNC-M [12]. Поэтому высокий уровень оптимизации программного кода новой реализации ОПТ, не

говорит об алгоритмическом его превосходстве над другими тестируемыми алгоритмами.

Таблица 1. Сравнение времени работы реализаций ОПТ, PUNC и RUNC-M

m	*n*	Средняя мощность *P(L)*	Время работы, сек				
			ОПТ1	ОПТ (новая реализация)	ОПТ [12]	PUNC [12]	RUNC-M [12]
10	100	135 491	0.03	0.04	0.06	0.05	0.05
10	200	3 195 275	0.28	0.57	0.93	0.64	1.03
10	300	19 411 998	1.96	3.36	5.61	3.82	6.65
10	400	84 181 874	8.12	14.76	23.54	16.34	32.28
20	50	41 604	0.02	0.03	0.04	0.03	0.03
20	100	1 224 641	0.14	0.29	0.50	0.38	0.44
20	150	10 352 239	1.17	2.37	4.02	3.07	3.94
20	200	48 177 652	5.26	10.98	18.27	13.96	19.76
30	50	105 534	0.03	0.05	0.08	0.06	0.06
30	100	4 625 484	0.54	1.28	2.24	1.78	1.89
30	150	44 376 844	5.48	11.74	20.18	16.47	19.46
30	200	232 206 320	26.97	53.40	104.01	84.05	110.16
40	50	198 939	0.04	0.12	0.16	0.12	0.10
40	100	10 635 194	1.59	3.47	5.76	4.79	4.76
40	150	113 767 855	16.11	34.16	58.37	50.15	55.89
40	200	664 505 102	85.58	194.06	333.06	294.22	350.84
100	40	341 229	0.11	0.36	0.40	0.34	0.22
200	40	1 156 362	0.49	1.89	1.68	1.69	0.85
300	40	2 243 626	1.22	5.11	3.94	4.35	1.84
400	40	3 741 902	2.64	10.86	6.95	8.62	3.28
70	50	717 594	0.17	0.48	0.66	0.53	0.42
150	50	3 609 021	1.15	4.10	4.26	4.07	2.36
230	50	8 259 075	3.34	13.05	11.85	12.53	6.21
310	50	15 235 401	7.23	31.58	24.15	28.67	12.21
30	30	8 014	0.01	0.02	0.08	0.03	0.02
60	60	1 549 823	0.29	0.76	1.14	0.95	0.73
90	90	61 999 773	11.69	32.20	49.03	46.23	35.05

120	120	1 040 345 733	224.71	619.16	833.43	944.00	669.79

4.4. Тестирование алгоритма ОПТ1

Результаты тестирования представлены в таблицах 1 и 2. Эксперименты проводились на случайных булевых матрицах с равновероятным появлением нулей и единиц. Для каждой пары (m, n) обсчитывалось по 20 случайных матриц (по 10 для матриц 30×200, 40×200, 120×120). В таблице 1 приведено среднее время работы различных реализаций алгоритмов при различных m и n (под средним понимается медиана). В таблице 2 приведены медианные значения следующих характеристик дерева решений, которое строит алгоритм ОПТ и ОПТ1: число неприводимых покрытий (мощность $P(L)$), число вершин и доля «лишних» шагов. Напомним, что на каждом шаге либо строится неприводимое покрытие, либо он является «лишним». Число всех шагов совпадает с числом висячих вершин.

Теперь выводы, которые можно сделать об алгоритме ОПТ1.

1. Согласно таблице 1, Настоящая реализация алгоритм ОПТ1 является безоговорочным лидером по скорости счета среди рассмотренных реализаций при указанных входных данных.
2. При $m \gg n$, при увеличении m преимущество алгоритма ОПТ1 уменьшается по сравнению с RUNC-M. Вероятно, это связано с удалением охватывающих строк в алгоритме ОПТ1; эта операция имеет квадратичную сложность относительно m. В RUNC-M такие строки не удаляются.
3. Дерево решений алгоритма ОПТ1 меньше в 1.2 – 3 раза , чем у алгоритма ОПТ. При $m \gg n$ дерево решений алгоритма ОПТ1 в 3 раза меньше, чем у алгоритма ОПТ, что делает ОПТ1 эффективным и при таких значениях m и n.
4. Согласно таблице 2, алгоритм ОПТ1 делает в 2 – 4 раза меньше «лишних» шагов, чем ОПТ. Наименьшая доля лишних шагов достигается при $m \ll n$.

Таблица 2. Сравнение характеристик деревьев решений алгоритмов ОПТ и ОПТ1

m	*n*	*Средняя мощность P(L)*	*Среднее число вершин дерева решений*		*Средняя доля лишних шагов, %*	
			ОПТ	*ОПТ1*	*ОПТ*	*ОПТ1*
10	*100*	*121 969*	*199 045*	140 321	35.13	10.91
10	*200*	*2 763 765*	*3 858 190*	3 030 407	26.07	7.77
10	*300*	*20 180 914*	*27 555 887*	21 728 995	24.56	6.40
10	*400*	*85 956 913*	*111 384 937*	91 687 869	21.06	5.72
20	*50*	*38 365*	*81 903*	48 330	48.78	15.97
20	*100*	*1 240 648*	*2 296 202*	1 470 891	41.75	12.75
20	*150*	*10 578 786*	*18 478 673*	12 167 019	39.30	10.97
20	*200*	*48 248 021*	*78 725 486*	54 397 835	35.51	9.72
30	*50*	*97 725*	*246 889*	126 411	56.12	17.72
30	*100*	*4 360 905*	*8 800 988*	5 289 251	46.84	14.48
30	*150*	*46 250 538*	*87 564 137*	54 213 498	43.59	12.26
30	*200*	*210 152 255*	*367 418 001*	242 678 614	39.44	11.38
40	*50*	*191 928*	*516 251*	252 876	58.54	18.63
40	*100*	*11 362 309*	*24 650 621*	14 022 953	50.18	15.41
40	*150*	*122 799 455*	*244 507 451*	146 329 566	46.12	13.36
40	*200*	*653 063 324*	*1 216 378 709*	761 760 796	42.81	12.04
100	*40*	*342 403*	*1 273 664*	511 722	69.73	25.04
200	*40*	*1 134 390*	*5 001 514*	1 792 261	73.69	27.99
300	*40*	*2 205 125*	*10 712 657*	3 625 961	75.94	30.19
400	*40*	*3 589 694*	*19 745 798*	6 084 218	78.59	31.60
70	*50*	*745 507*	*2 362 391*	1 043 615	64.37	22.06
150	*50*	*3 649 464*	*13 979 487*	5 465 264	70.22	25.88
230	*50*	*8 205 391*	*35 036 101*	12 758 745	72.97	27.69
310	*50*	*15 235 401*	*72 258 828*	24 070 769	75.65	28.78
30	*30*	*7 808*	*22 331*	10 838	60.36	20.72
60	*60*	*1 604 744*	*4 596 740*	2 147 898	61.17	19.86
90	*90*	*62 891 964*	*169 951 694*	83 762 634	59.18	20.08
120	*120*	*1 040 345 733*	*2 847 483 647*	1 383 769 920	58.54	20.17

5. Получение оценок для объёмов вычислительных подзадач

В данном разделе три подраздела. В подразделах 5.1 и 5.2 описаны способы оценки объёмов подзадач, или величин $v_j(L)$, используемые соответственно в B-схеме и S-схеме. В подразделе 5.3 предложен способ предобработки исходной матрицы, позволяющий сделать распределение величин $v_j(L)$ более равномерным.

В B-схеме на пространстве элементарных событий $\Omega_1 = \{(L,H)|L \in M_{mn},\ H \in P(L)\}$ с равновероятными исходами вводится случайная величина $\eta(L,H)$, равная j, если $H \in P_j(L),\ j \in \{1,\dots,n\}$. При помощи критерия Хи-квадрат проверяется гипотеза о виде распределения $H_0: f(j) = \psi_{\alpha\beta}(j)$, где $f(j)$ – вероятность события $\eta(L,H) = j$, а $\psi_{\alpha\beta}(j)$ – функция вероятности бета-биномиального распределения с параметрами α и β, которые оцениваются при помощи метода максимального правдоподобия по случайной выборке. B-схема использует эти параметрические оценки для приближения искомой величины $v_j(L)$.

В S-схеме используется другое пространство элементарных событий. Пусть L – матрица размера $m \times n$, $r \in \{1,\dots,m\}$ и W_m^r – множество всех подмножеств множества $\{1,\dots,m\}$ мощности r. Через L^w, $w \in W_m^r$, обозначим подматрицу матрицы L, составленную из строк матрицы L с номерами из w. Вводится пространство элементарных событий $\Omega_2^r = \{(L^w,H)|w \in W_m^r,\ H \in P(L^w)\}$ с равновероятными исходами и случайная величина $\eta_r(L^w,H)$, равная j, $j \in \{1,\dots,n\}$, если $H \in P_j(L^w)$. При помощи критерия Хи-квадрат проверяется гипотеза о виде распределения $H_0: f_r(j) = v_j(L)$, где $f_r(j)$ – вероятность события $\eta_r(L^w,H) = j$. S-схема использует эмпирические оценки для $f_r(j)$ как приближение величины $v_j(L)$.

5.1. Оценки для подзадач, используемые в B-схеме

В качестве пространства элементарных событий Ω_1 рассмотрим множество всевозможных пар (L, H), $L \in M_{mn}$, $H \in P(L)$. Положим вероятность события (L, H) равной $|\Omega_1|^{-1}$.

Введём на Ω_1 случайную величину $\eta(L, H)$, равную j, если $H \in P_j(L)$, $j \in \{1, \dots, n\}$. Через $f(j)$ обозначим вероятность события $\eta(L, H) = j$.

Говорят, что случайная величина ξ имеет бета-биномиальное распределение $B(\alpha, \beta, r)$, если она принимает значения q из множества $\{0,1, \dots, r\}$ с вероятностью

$$\varphi(q; \alpha, \beta, r) = \binom{r}{q} \frac{B(q + \alpha, r - q + \beta)}{B(\alpha, \beta)},$$

где B – бета-функция, а $\alpha > 0$, $\beta > 0$ и $r \in \mathbb{N}$ – параметры распределения. Обозначим

$$\psi_{\alpha\beta}(j) = \varphi(j - 1; \alpha, \beta, n - 1), \qquad j \in J_n.$$

Проведём статистический эксперимент. Пусть $x = (x_1, \dots, x_N)$ – выборка из распределения $f(j)$. Проверим статистическую гипотезу $H_0 = \{f(j) = \psi_{\alpha\beta}(j)\}$ о виде распределения. В случае, когда гипотеза верна, случайная величина $\eta - 1$ имеет распределение $B(\alpha, \beta, n - 1)$.

Для проверки гипотезы H_0 используется видоизменённый критерий Хи-квадрат. Статистика $Z(x)$ этого критерия может быть определена как

$$N \sum_{j=1}^{n} \frac{\left(f^*(j) - \psi_{\alpha\beta}(j)\right)^2}{\psi_{\alpha\beta}(j)},$$

где $f^*(j)$ – доля элементов выборки $x = (x_1, \dots, x_N)$, равных j. При вычислении $\psi_{\alpha\beta}(j)$ значений параметров α и β принимаются равными оценкам максимального правдоподобия. На рисунке 3 приведен пример графиков $f^*(j)$ и $\psi_{\alpha\beta}(j)$ для конкретной реализации выборки x для матриц из $M_{30,120}$.

Пусть задан уровень значимости γ и пусть $\chi^2_{\delta,r}$ – квантиль уровня δ распределения Хи-квадрат с r степенями свободы. Гипотеза H_0 отвергается тогда и только тогда, когда $Z(x) > \chi^2_{1-\gamma,n-3}$ (напомним, что по выборке x оцениваются два параметра функции $\psi_{\alpha\beta}(j)$). Следуя этому критерию, можно ошибочно отклонить гипотезу H_0, когда она верна, с вероятностью, приближённой равной γ.

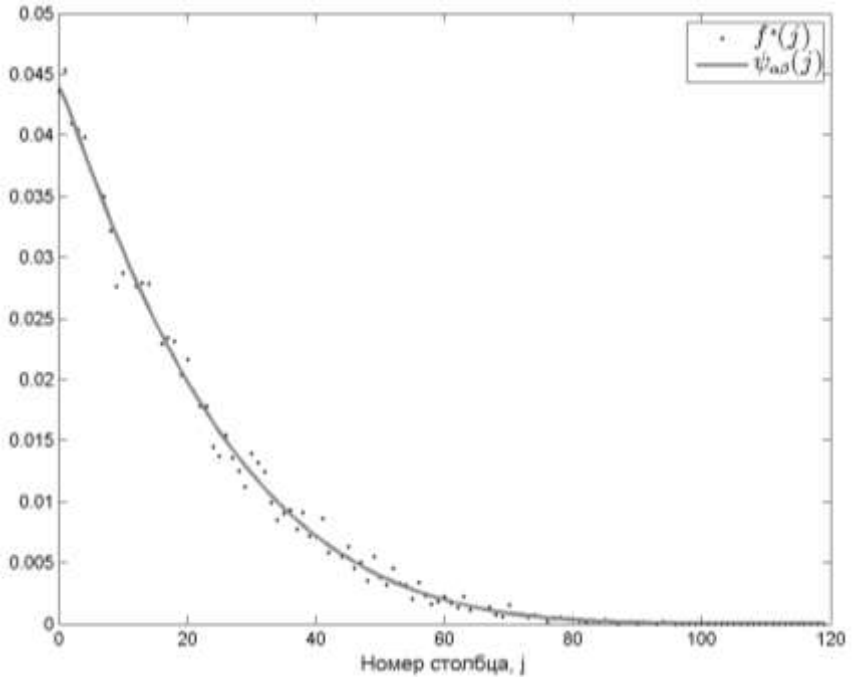


Рис. 3. Графики f(j) и ψ$_{αβ}$(j) для m=30, n=120*

Достигнутым уровнем значимости критерия Хи-квадрат называется величина $\gamma^*(x) = 1 - \chi^2_{n-3}(Z(x))$, где $\chi^2_{n-3}(.)$ – функция распределения Хи-квадрат с $n-3$ степенями свободы. Близость значения $\gamma^*(x)$ к 0 говорит о том, что гипотезу H_0, вероятнее всего, следует отклонить. Возникает вопрос, как построить выборку $x = (x_1, \dots, x_N)$ из распределения $f(j)$. Предлагается следующий способ. Построим t случайных матриц из M_{mn}: $L_1, \dots, L_t$. Из совокупности множеств $P(L_k)$, $k \in J_t$, выберем $N = tu$ пар матрица-покрытие. Возьмём N значений случайной величины η на этих парах и сформируем выборку.

Проведены эксперименты с выборками, построенными по матрицам размера $m \times n$ при $m = 20,\ 35,\ 50,\ 65,\ 80$ и $n = 20,\ 35,\ 50,\ 65,\ 80$. Для получения выборки

полагалось $t = 200,\ u = 10$. В таблицах 3 – 5 приведены полученные в ходе экспериментов значений достигнутого уровня значимости $\gamma^*(x)$ и оценок максимального правдоподобия α и β.

Проведённые эксперименты позволяют сделать следующие выводы.

- При $n = 20$ гипотезу H_0 следует отклонить. Возможной причиной этого является нерепрезентативность выборки, связанная с небольшим числом неприводимых покрытий у матриц при $n = 20$.
- Для конфигураций 20×35 и 20×50 достигнутый уровень значимости относительно невелик ввиду небольшого числа неприводимых покрытий у матриц данного размера.
- Матрицы размера 80×80, как правило, имеют большое число неприводимых покрытий (порядка 10^7), что также мешает получению репрезентативной выборки ограниченного объема используемым в работе способом. Достигнутый уровень значимости в этом случае достаточно мал.
- В остальных случаях гипотезу H_0 можно принять с большой уверенностью.

Таблица 3. Достигнутый уровень значимости $\gamma^*(x)$ для B-схемы

m \ n	20	35	50	65	80
20	0.001	0.106	0.326	0.502	0.751
35	0.001	0.517	0.992	0.842	0.974
50	0.001	0.919	0.515	0.647	0.983
65	0.001	0.638	0.719	0.327	0.556
80	0.001	0.303	0.969	0.79	0.252

Таблица 4. Оценка параметра α для B-схемы

m \ n	20	35	50	65	80
20	*1.193*	*1.048*	*0.988*	*0.994*	*0.959*

35	*1.086*	*1.006*	*1.005*	*0.957*	*0.993*
50	*1.051*	*1.049*	*0.975*	*1.01*	*0.993*
65	*0.994*	*1.015*	*0.966*	*1.001*	*0.985*
80	*0.978*	*0.959*	*0.996*	*0.985*	*1.011*

Таблица 5. Оценка параметра β для B-схемы

m \ n	20	35	50	65	80
20	*6.062*	*5.118*	*4.762*	*4.941*	*4.75*
35	*6.694*	*5.901*	*5.649*	*5.427*	*5.51*
50	*7.283*	*6.51*	*6.12*	*6.26*	*6.14*
65	*7.013*	*6.887*	*6.432*	*6.633*	*6.491*
80	*7.727*	*6.887*	*6.938*	*6.962*	*6.69*

5.2.Оценки для подзадач, используемые в S-схеме

Пусть $L \in M_{mn}$ и $r \leq m$.. Через W_m^r обозначим множество всех подмножеств мощности r множества $\{1, \dots, m\}$. Пусть $w \in W_m^r$, тогда через L^w обозначим подматрицу матрицы L, составленную из строк матрицы L с номерами из w.

Пусть $\Omega_2^r = \{(L^w, H) | w \in W_m^r,\ H \in P(L^w)\}$ – пространство элементарных событий с равновероятными исходами. На указанном пространстве определим случайную величину $\eta_r(L^w, H)$, которая равна j, $j \in J_n$, если $H \in P_j(L^w)$. Через $f_r(j)$ обозначим вероятность события $\eta_r(L^w, H) = j$.

Предлагается использовать величины $f_r(j)$ как оценки для величин $\nu_j(L)$. Встаёт вопрос, как правильно выбрать величину r. С одной стороны, эта величина должна иметь как можно меньшее значение, чтобы время получения оценок было относительно невелико. С другой стороны, оценки должны быть достоверными.

Пусть $x = (x_1, \dots, x_N)$ – выборка из распределения $f_r(j)$. Для проверки статистической гипотезы $H_0 = \{f_r(j) = \nu_j(L)\}$ о виде распределения предлагается использовать

видоизменённый критерий Хи-квадрат, как и в разделе 5.1. Статистика $Z(x)$ этого критерия может быть определена как

$$N\sum_{j=1}^{n}\frac{\left(f_r^*(j)-\nu_j(L)\right)^2}{\nu_j(L)},$$

где $f^*(j)$ – доля элементов выборки $x=(x_1,\dots,x_N)$, равных j.

Достигнутым уровнем значимости критерия Хи-квадрат называется величина $\gamma^*(x)=1-\chi^2_{n-1}\big(Z(x)\big)$, где $\chi^2_{n-1}(.)$ – функция распределения Хи-квадрат с $n-1$ степенями свободы. Близость значения $\gamma^*(x)$ к 0 говорит о том, что гипотезу H_0, вероятнее всего, следует отклонить. Для получения выборки $x=(x_1,\dots,x_N)$ из распределения $f(j)$ построим t случайных подматрицы L^w матрицы L размера $r\times n$. Из совокупности множеств $P(L_k)$, $k\in J_t$, выберем N пар матрица-покрытие. Возьмём N значений случайной величины η_r на парах матрица-покрытие и сформируем выборку.

Проведём эксперимент. Для каждой из конфигураций $\{30\times 120,\ 40\times 120, 50\times 100,\ 70\times 70\}$ берётся по 20 случайных матриц соответствующего размера. Для каждой матрицы формируется выборка $x=(x_1,\dots,x_N)$ из распределения $f_r(j)$, где $N=1000$ и $t=20$. В таблицах 6 и 7 приведены медианные значений статистики $Z(x)$ и достигаемых уровней значимости $\gamma^*(x)$. На рисунке 4 приведён график зависимости $Z(x)$ от r. На рисунке 5 приведены графики величин $\nu_j(L)$ и $f_r^*(j)$.

Согласно таблице 7, минимальное значение r, при котором достигнутый уровень значимости $\gamma^*(x)$ не является пренебрежимо малым, как правило, равняется $m/2$. По рисунку 4 на примере конфигурации 30×150 можно заметить, что имеет место «фазовый переход» при $r = 15$: при пересечении этой точки функция $Z(x)$ начинает медленее уменьшаться. Это говорит о том, что дальнейшее увеличение r не принесёт существенного выигрыша в приближении $\nu_j(L)$.

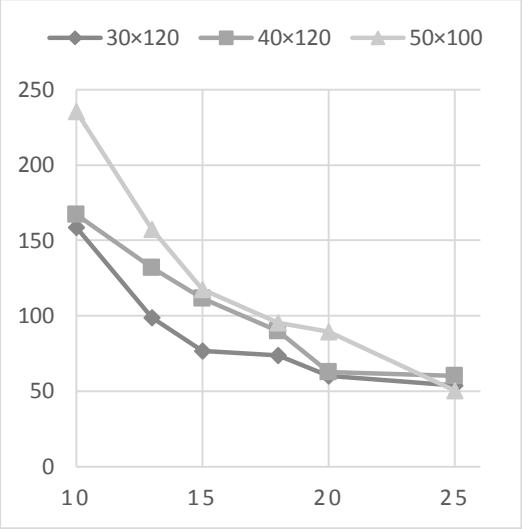


Рис. 4. Зависимость Z(x) от r для различных m и n

Таблица 6. Значения статистики Z(x) критерия Хи-квадрат для различных комбинаций m, n и r

r \ m×n	*30×120*	*40×120*	*50×100*	*70×70*
10	158.7337	167.0505	235.4325	381.8272
13	98.6796	132.0449	157.4014	233.7598
15	76.6111	111.5510	117.3677	186.5517
18	73.8490	89.9256	95.6307	146.5449
20	60.1580	62.5984	89.4963	130.8454
25	53.8773	60.2040	50.2776	84.5474
30	-	-	-	68.2291
35	-	-	-	53.7886

Таблица 7. Достигнутые уровни значимости γ(x) для различных комбинаций m, n и r*

r \ m×n	*30×120*	*40×120*	*50×100*	*70×70*
10	$<10^{-4}$	$<10^{-4}$	$<10^{-4}$	$<10^{-4}$
13	$<10^{-4}$	$<10^{-4}$	$<10^{-4}$	$<10^{-4}$
15	**0.0134**	$<10^{-4}$	$<10^{-4}$	$<10^{-4}$
18	0.0280	0.0002	$<10^{-4}$	$<10^{-4}$
20	0.0815	**0.0546**	$<10^{-4}$	$<10^{-4}$
25	0.3150	0.0876	**0.1382**	$<10^{-4}$

30	-	-	-	0.0001
35	-	-	-	**0.0478**

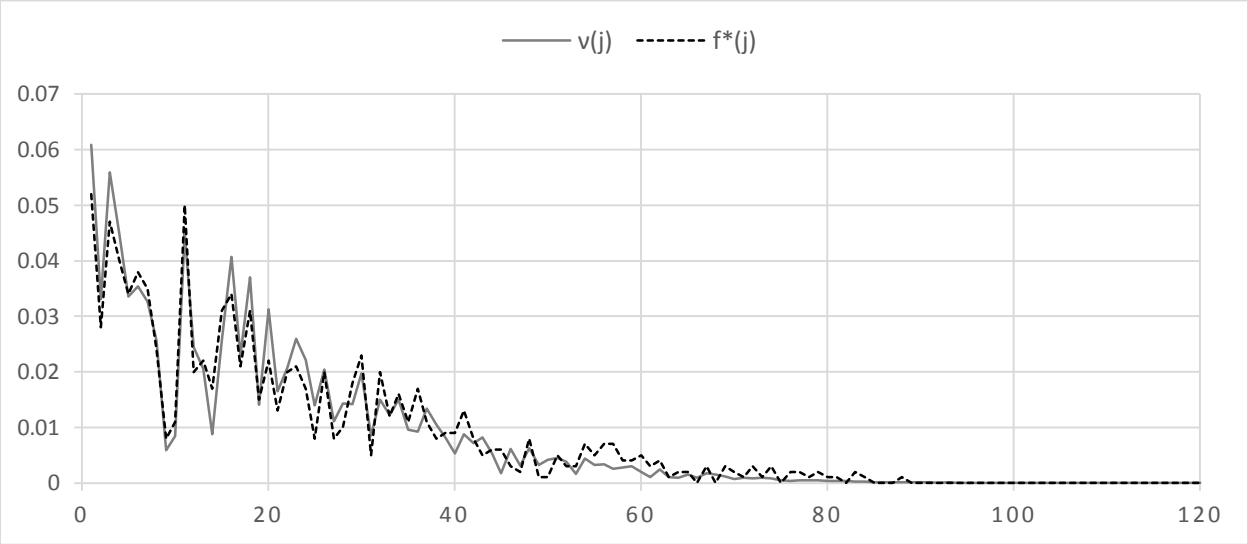


Рис. 5. Графики $v_j(L)$ и $f_r^*(r)$ как величин, зависящих от j, при m=30, n=120, r=15

5.3.Предобработка исходной матрицы

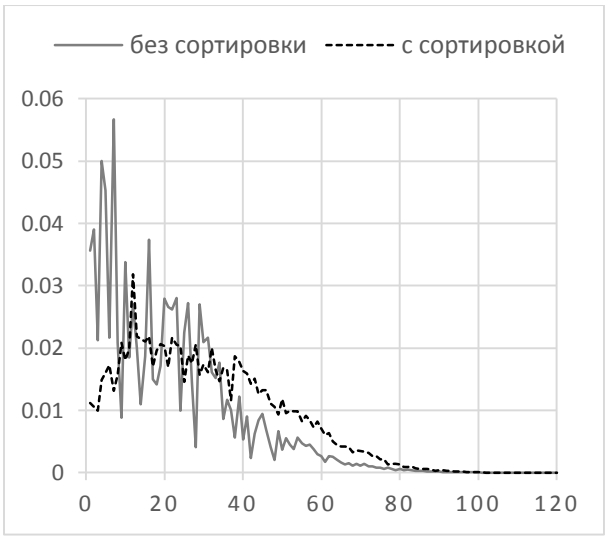
Напомним, что $\nu_j(L)$ есть доля j-неприводимых покрытий матрицы L. Обозначим $\nu(L) = \left(\nu_1(L), \dots, \nu_n(L)\right)$ и $c(L) = \left(c_1(L), \dots, c_n(L)\right)$, где $c_j(L) = \sum_{u=1}^{j} \nu_u(L)$. Пусть $L \in M_{mn}$. Упорядочим столбцы матрицы L в порядке убывания величин $w_j = \sum_{i=1}^{m} a_{ij}$. Обозначим получившуюся матрицу через L'.

Введём следующие величины:

- $V_1(L)$, равная доле j, для которых $\nu_j(L) \geq 0.5 \max \nu_j(L)$;
- $V_2(L) = \sqrt{\sum_{j=1}^{n}\left(c_j(L) - j/n\right)^2}$, равная среднеквадратическому отклонению вектора $c(L)$ от вектора $\left(\frac{1}{n}, \frac{2}{n}, \dots, \frac{n-1}{n}, \frac{n}{n}\right)$.

На рисунках 6 и 7 продемонстрированы графики для $\nu(L)$, $\nu(L')$ и $c(L)$, $c(L')$. Очевидно, что векторы $\nu(L)$ и $\nu(L')$ отличаются не только порядком компонент. Согласно графикам, величина $\mathrm{V}_1(L) \approx 20$, в то время, как $V_1(L') \approx 40$. Таким образом, величина

$V_1(L)$ характеризует размер «плато» при максимуме $\nu_j(L)$. Величина $V_2(L)$ показывает, насколько график $c(L)$ близок к линейной функции.



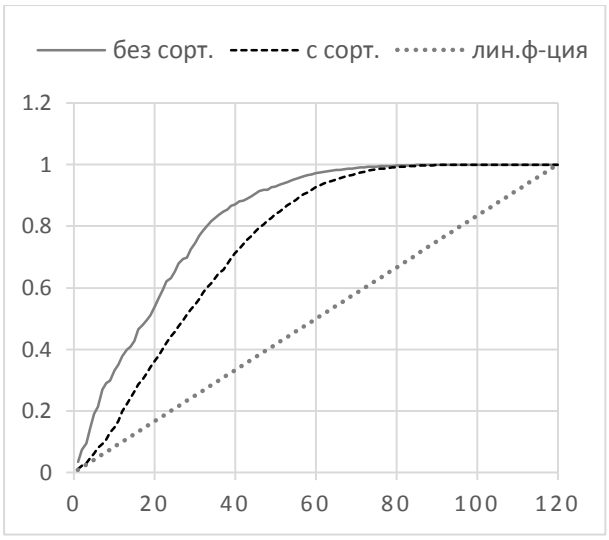


Рис. 6. v(L) как функция от j при m=30, n=120 *Рис. 7. c(L) как функция от j при m=30, n=120*

В таблице 8 отражены статистические характеристики величин $\delta V_i = V_i(L') - V_i(L), i \in \{1,2\}$, где L' – матрица с отсортированными столбцами. Для экспериментов бралось по 20 случайных матриц каждой конфигурации. Через $p(1)$ обозначим вероятность появления единицы на позиции (i,j) при генерации матрицы $L = (a_{ij}) \in M_{mn}$.

Таблица 8. Влияние сортировки столбцов матрицы L на вектор $\nu(L)$

m	*n*	*p(1)*	*min* δV_1	медиана δV_1	*max* δV_1	*min* δV_2	медиана δV_2	*max* δV_2
30	120	0.5	0.192	0.267	0.300	-0.107	-0.088	-0.073
50	50	0.5	0.040	0.120	0.180	-0.050	-0.036	-0.027
120	30	0.5	-0.033	0.033	0.067	-0.016	-0.007	0.024
20	40	0.2	-0.060	0.050	0.120	-0.066	-0.019	0.018
30	120	0.8	0.158	0.308	0.400	-0.128	-0.107	-0.081

Ясно, что наилучшие результаты при сортировке столбцов достигаются, когда n значительно больше m. В обратном случае наблюдаются наихудшие результаты: сортировка столбцов перестаёт предсказуемым образом влиять на $\nu(L)$. На рисунках 6 – 9 в виде графиков представлены «типичные» значения ν_j и c_j в виде графиков для двух

конфигураций матриц: в наилучшем ($m = 30, n = 120$) и наихудшем случаях ($m = 120, n = 30$).

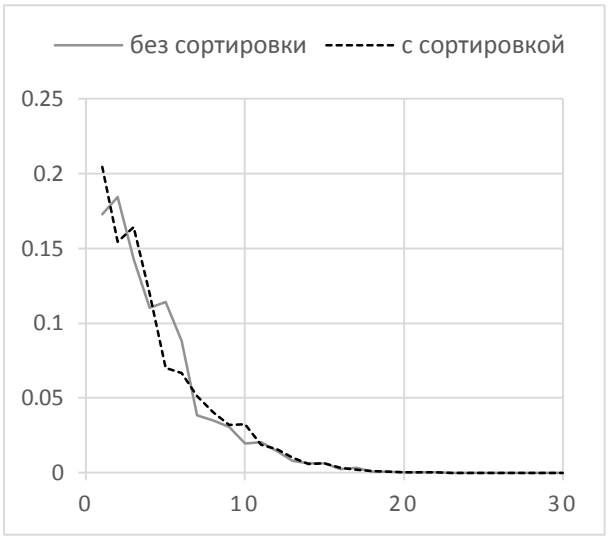


Рис. 8. v(L) как функция от j при m=120, n=30

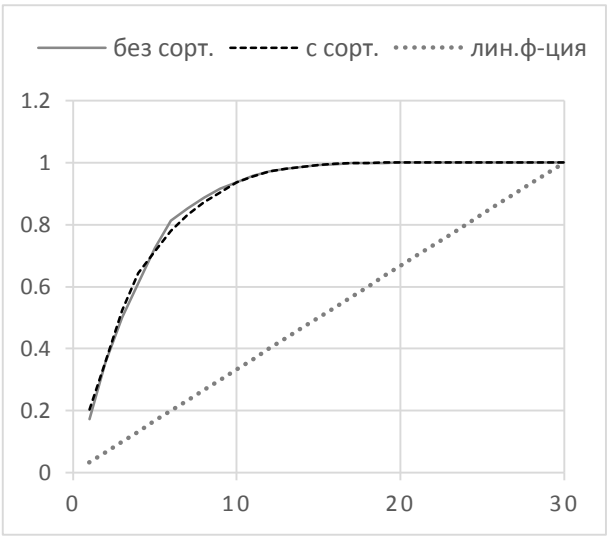


Рис. 9. c(L) как функция от j при m=120, n=30

Теперь изучим, как влияет сортировка столбцов матрицы L на оценки, полученные в разделе 5.2. Проведём аналогичные эксперименты. Результаты представлены в таблицах 9 и 10.

Для матриц размера 40×120 удалось уменьшить значение r с 20 до 18, а для 40×120 незначительно уменьшить значение статистики $Z(x)$. Как показано в этом разделе, при $m \ll n$ сортировка столбцов матрицы приближает распределение величин $\nu_j(L)$ к равномерному. Видимо, благодаря этому результаты экспериментов улучшились.

Для матриц 50×100 и 70×70 получены худшие результаты, чем в подразделе 5.2.

Таблица 9. Значения статистики Z(x) критерия Хи-квадрат для различных комбинаций m, n и r, когда столбцы матрицы L отсортированы по убыванию числа единиц

r \ m×n	*30×120*	*40×120*	*50×100*	*70×70*
10	104.8204	137.7527	190.0770	307.9484
13	96.5271	110.6625	152.2002	254.9508
15	79.4265	99.5938	122.7795	170.3230
18	77.3518	80.3226	107.9616	144.1791
20	69.6391	73.3947	100.1684	127.9294

25	60.4672	64.6044	69.0567	92.4512
30	-	-	-	75.1030
35	-	-	-	69.6064

Таблица 10. Достигнутые уровни значимости γ(x) для различных комбинаций m, n и r, когда столбцы матрицы L отсортированы по убыванию числа единиц*

r \ m×n	*30×120*	*40×120*	*50×100*	*70×70*
10	0.0006	<10-4	<10-4	<10-4
13	0.0026	<10-4	<10-4	<10-4
15	**0.0329**	0.0004	<10-4	<10-4
18	0.0672	**0.0215**	<10-4	<10-4
20	0.1611	0.0526	<10-4	<10-4
25	0.5021	0.1843	**0.0175**	<10-4
30	-	-	-	<10-4
35	-	-	-	0.0005

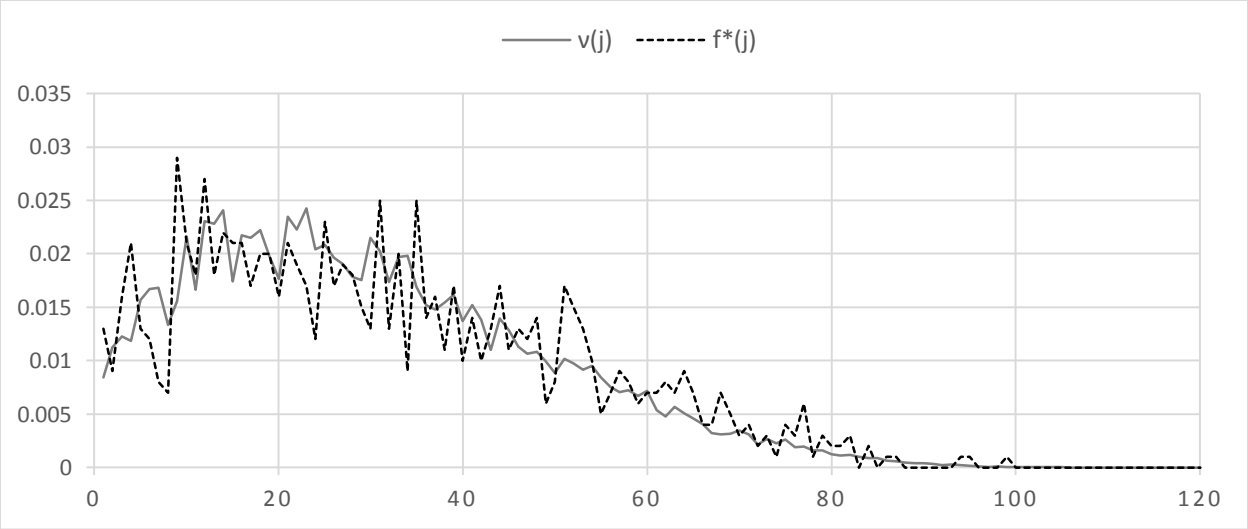


Рис. 10. Графики $v_j(L)$ и $f_r^*(r)$ как величин, зависящих от j, при m=30, n=120, r=15, когда столбцы матрицы L отсортированы по убыванию числа единиц

6. Распределение вычислительных заданий между процессорами

Пусть $L \in M_{mn}$ и пусть дано $p \leq n$ процессоров. Каждому из множеств $P_j(L)$ сопоставим процессор с номером N_j^p, который должен это множество построить. Вектор

$N^p = \left(N_1^p, \dots, N_n^p\right)$ назовём расписанием. Уровнем загрузки k-го процессора назовём величину

$$\sigma_k^0(L, N^p) = \sum_{j \in J_n : N_j^p = k} \nu_j(L).$$

Для эффективного распределения вычислительных заданий между процессорами, требуется решить следующую задачу минимизации уровня загрузки процессоров

$$\sigma_0(L, N^p) = \max_{k \in J_p} \sigma_k^0(L, N^p) \to \min_{N^p} . \quad (1)$$

Ниже приведено описание процедуры 3, которая ищет приближённое решение описанной выше задачи при помощи жадного алгоритма. На вход этой процедуры подаётся число процессоров, число столбцов матрицы L и вектор $\tilde{\nu} = (\tilde{\nu}_1, \dots, \tilde{\nu}_n)$, состоящий из оценок $\tilde{\nu}_j$ для величин $\nu_j(L)$. B-схема, в качестве оценок $\tilde{\nu}_j$, использует значения функции вероятности $\psi_{\alpha\beta}(j)$ бета-биномиального распределения, где α и β получены методом максимального правдоподобия. В S-схеме же полагается $\tilde{\nu}_j = \hat{\nu}_j(L, r)$.

В данном случае жадная стратегия является достаточно эффективной, потому что число малых значений $\nu_j(L)$, как правило, велико (более половины).

***Процедура** 3. $N^p = DistributeTasks(p, n, \tilde{\nu})$*

1. FOR $k \in \{1, \dots, p\}$
2. $\quad \sigma_k = 0$
3. ENDFOR
4. FOR $j \in \{1, \dots, n\}$
5. $\quad k_0 = \arg\min_{k \in J_p} \sigma_k$
6. $\quad N_j^p = k_0$
7. $\quad \sigma_k = \sigma_k + \tilde{\nu}_j$
8. ENDFOR

Фиксируем матрицу $L \in M_{mn}$. Тогда распределение вычислительных заданий на первом уровне дерева решений, которое строит алгоритм ОПТ, накладывает ограничения на число процессоров, при котором предложенные схемы распараллеливания эффективны (B-схема и П-схема). При небольших p задача минимизации (1) имеет решение, при котором $\sigma_0(L, N^p) \approx 1/p$. При увеличении p, когда это число превосходит некоторое p^*, величина $\sigma_0(L, N^p)$ перестаёт быть близкой к $1/p$, что повлечёт за собой неравномерную загрузку процессоров.

7. Тестирование разработанных параллельных алгоритмов дуализации

В данном разделе приведены

- описание среды вычислений (суперкомпьютер IBM Blue Gene/P);
- описание исследуемых показателей работы параллельных алгоритмов;
- результаты сравнения разработанных B-схемы, S-схемы и тривиальной U-схемы на матрицах размера 65×80, 80×65 и 80×80;
- результаты тестирования S-схемы на матрицах $L \in M_{mn}$, где $m \ll n$, например, 30×200 и 40×200 (на таких конфигурациях вычисление параметров бета-биномиального распределения в B-схеме является вычислительно сложным).

Отметим, что результаты тестирования проводятся только для алгоритма ОПТ1, описанном в разделе 3, так как он является лидером по скорости счёта.

7.1. Среда тестирования

Тестирование проводилось на суперкомпьютере IBM Blue Gene/P. Этот суперкомпьютер располагается в МГУ им. Ломоносова в здании факультета Вычислительной математики и кибернетики и является массивно-параллельной вычислительной система, которая состоит из двух стоек, включающих 8192 процессорных ядер (2 x 1024 четырехъядерных вычислительных узлов), с пиковой производительностью

27,9 терафлопс (27,8528 триллионов операций с плавающей точкой в секунду). Каждый вычислительный узел включает в себя четырехъядерный процессор PowerPC 450 (850 MHz), 2 GB общей памяти и сетевые интерфейсы.

При запуске вычислительных заданий использовался режим виртуальных вычислительных узлов (VN-режим). В этом режиме на каждом вычислительном узле запущено четыре MPI-процесса, которые делят между собой доступные ресурсы (в первую очередь память и сеть трехмерного тора), причем за разделение ресурсов и возможность их независимого использования отвечает ядро вычислительного узла. Каждому из MPI-процессов доступно по 472 MB оперативной памяти.

7.2. Показатели эффективности параллельного алгоритма

Через p обозначим число процессоров, через $T_k(p)$ – время (в секундах) работы k-го процессора параллельной версии алгоритма при использовании p процессоров. Пусть $T(p) = \max_k T_k(p)$ и $T_\Sigma(p) = \sum_k T_k(p)$. Исследуются три показателя:

- ускорение алгоритма $S(p) = T(1)/T(p)$;
- равномерность загрузки процессоров $E(p) = S(p)/p$;
- достигнутый уровень загрузки $\sigma(p) = T(p)/T_\Sigma(p)$.

Ускорение, соответствующее линейной функции $S(p) = p$ при $p \geq 1$, является практически максимальным. В некоторых случаях может возникать суперскалярное ускорение, когда $S(p)$ растёт быстрее, чем p; оно может достигаться за счёт более эффективного использования процессорного кэша.

Преобразуем второй показатель к следующему виду: $E(p) = (T(1)/p)/T(p)$ – в этой формуле $T(1)/p$ соответствует равномерной загрузке процессоров в идеальном случае, а $T(p)$ – максимальному времени работы среди процессоров, что соответствует всему времени работы.

Показатель $\sigma(p)$ является аналогом показателя уровня загрузки процессоров $\sigma_0(L, N^p)$, определённого в разделе 6.

7.3. Сравнение разработанных схем распараллеливания

Опишем тривиальную схему распараллеливания асимптотически оптимального алгоритма ОПТ (U-схему). По аналогии с B-схемой и M-схемой, U-схема основана на использовании расписания $N^p = \left(N_1^p, \dots, N_n^p\right)$, где $N_j^p = \lceil pj/n \rceil$, $j \in J_n$, и $\lceil x \rceil$ равно ближайшему целому, не меньшему x. Другими словами, первые примерно n/p столбцов должны быть обработаны первым процессором, следующие n/p – вторым и так далее.

В B-схеме не предполагается вычисление оценок для параметров α и β функции $\psi_{\alpha\beta}(j)$ при работе параллельного алгоритма, так как они определяются только размерами матрицы. Кроме того, задача вычисления указанных оценок является сложной, потому что требуется построить множество всех неприводимых покрытий для нескольких десятков или сотен матриц данного размера (см. раздел 5.1).

В S-схеме, напротив, оценки $\hat{v}_j(L, r), j \in \{1, \dots, n\}$, подсчитываются во время работы параллельного алгоритма. Поэтому они учтены в $T(p)$. Значение параметра r, определяющего размер подматриц матрицы L, по которым строятся оценки $\hat{v}_j(L, r)$, полагалось равным 10.

Сравнение предложенных схем распараллеливания, в том числе и тривиальной, проводится на матрицах размера 65×80, 80×65 и 80×80. Матрицы больших конфигураций не рассматриваются, ввиду ограниченной применимости к ним B-схемы. Результаты представлены в таблицах 11 – 13. На рисунках 11 – 16 представлены графики функций $S(p)$ и $E(p)$ для всех трёх схем. На рисунке 23 продемонстрирован уровень загрузки $\sigma_k(p) = T_k(p)/T_\Sigma(p)$, $k \in \{1, \dots, p\}$, при $p = 16$.

Из указанных таблиц и графиков следует, что B-схема и S-схема демонстрируют практически одинаковое ускорение $S(p)$. При $p \leq 16$ это ускорение близко к линейному,

$S(p) \approx p$, и $E(p)$ достаточно велико, что говорит о высоком качестве распараллеливания. Тем не менее, при $p > 16$ параллельный алгоритм не работает быстрее. Это связано с тем, что распараллеливание происходит на первом ярусе дерева решений, которое строит алгоритм ОПТ. При таком подходе объёмы подзадач сильно различаются, поэтому их принципиально невозможно равномерно распределить между большим числом процессоров.

По времени работы S-схема незначительно уступает B-схеме (5-10%), так как она дополнительно вычисляет оценки $\hat{v}_j(L,r)$. По рисунку 23 видно, что в обеих разработанных схемах уровень загрузки в среднем одинаков, но существует несколько значительных отклонений от среднего (на 30-50%).

Отметим, что максимальное значение для ускорения, достигаемое B-схемой и S-схемой, практически совпадает с максимальным значением ускорения для U-схемы. Разница в том, при каких p оно достигается: для разработанных схем это число равно 16, а для U-схемы – 64.

Таблица 11. Сравнение схем распараллеливания при m=65, n=80

	B-схема			S-схема			U-схема		
p	*T(p)*	*σ₀(p)*	*σ(p)*	*T(p)*	*σ₀(p)*	*σ(p)*	*T(p)*	*σ₀(p)*	*σ(p)*
1	17.44	1.000	1.000	18.35	1.000	1.000	17.44	1.000	1.000
2	9.01	0.500	0.514	9.40	0.500	0.502	17.24	0.500	0.989
4	5.30	0.250	0.291	4.92	0.250	0.261	14.69	0.250	0.852
8	2.71	0.125	0.147	2.52	0.125	0.135	9.29	0.125	0.554
16	1.55	0.079	0.090	1.62	0.084	0.087	5.08	0.063	0.312
32	1.55	0.079	0.090	1.61	0.089	0.087	2.40	0.031	0.147
64	1.55	0.079	0.090	1.61	0.089	0.087	1.55	0.016	0.090

Таблица 12. Сравнение схем распараллеливания при m=80, n=65

	B-схема			S-схема			U-схема		
p	*T(p)*	*σ₀(p)*	*σ(p)*	*T(p)*	*σ₀(p)*	*σ(p)*	*T(p)*	*σ₀(p)*	*σ(p)*
1	26.17	1.000	1.000	26.29	1.000	1.000	26.17	1.000	1.000
2	13.72	0.500	0.514	13.87	0.500	0.507	25.73	0.500	0.987

4	7.17	0.250	0.271	7.01	0.250	0.255	21.58	0.250	0.848
8	3.87	0.125	0.140	3.79	0.126	0.137	13.98	0.125	0.572
16	2.83	0.102	0.114	3.13	0.086	0.123	8.25	0.063	0.340
32	2.83	0.102	0.114	3.13	0.092	0.123	4.31	0.031	0.184
64	2.83	0.102	0.114	3.13	0.092	0.123	2.83	0.016	0.114

Таблица 13. Сравнение схем распараллеливания при m=80, n=80

	B-схема			S-схема			U-схема		
p	*T(p)*	$\sigma_0(p)$	$\sigma(p)$	*T(p)*	$\sigma_0(p)$	$\sigma(p)$	*T(p)*	$\sigma_0(p)$	$\sigma(p)$
1	34.49	1.000	1.000	36.56	1.000	1.000	34.49	1.000	1.000
2	17.31	0.500	0.502	18.85	0.500	0.508	34.15	0.500	0.991
4	10.19	0.250	0.286	9.94	0.250	0.257	29.23	0.250	0.858
8	5.10	0.125	0.147	5.35	0.125	0.137	18.78	0.125	0.568
16	3.03	0.078	0.091	3.33	0.074	0.085	11.11	0.063	0.340
32	3.03	0.078	0.091	3.32	0.076	0.085	5.09	0.031	0.157
64	3.03	0.078	0.091	3.32	0.076	0.085	3.03	0.016	0.091

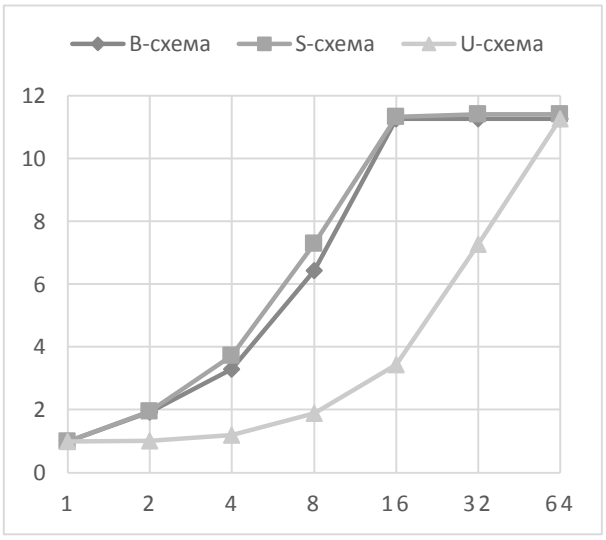


Рис. 11. График S(p) при m=65, n=80

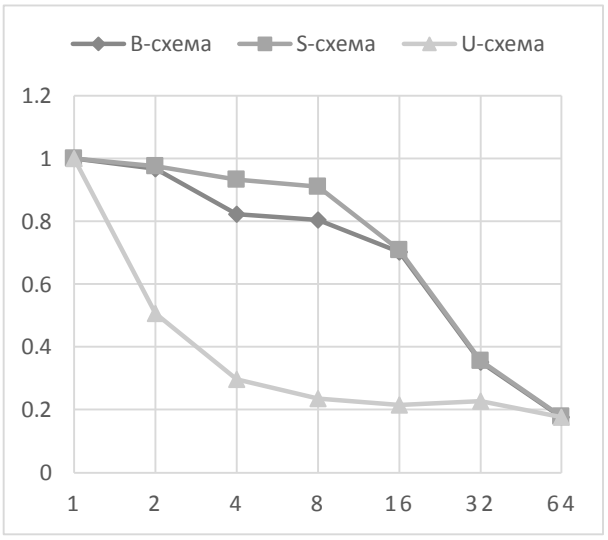


Рис. 12. График E(p) при m=65, n=80

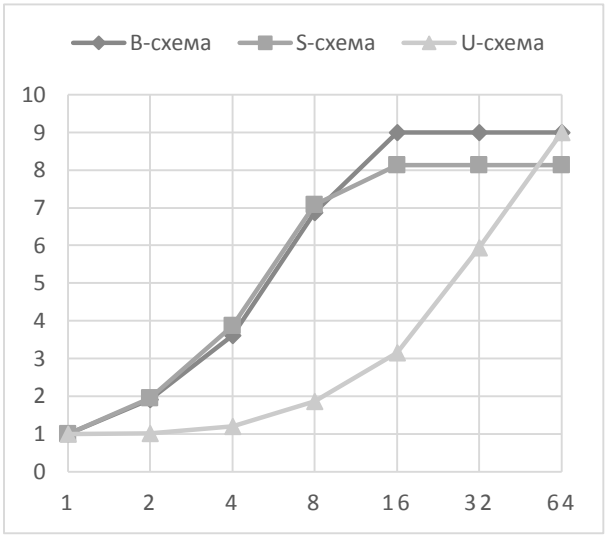


Рис. 13. График S(p) при m=80, n=65

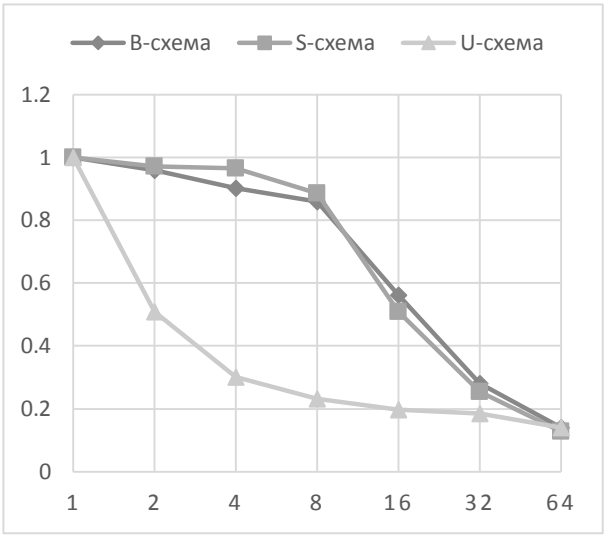


Рис. 14. График E(p) при m=80, n=65

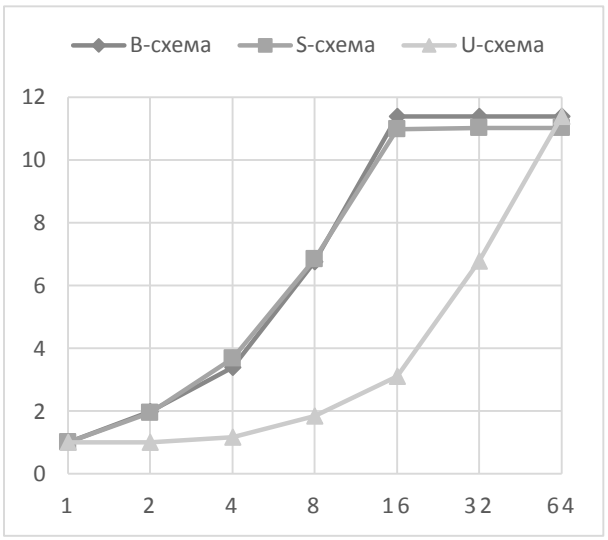


Рис. 15. График S(p) при m=80, n=80

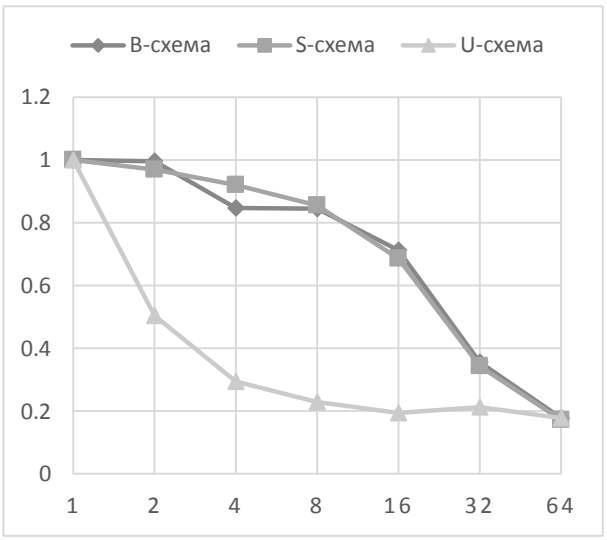


Рис. 16. График E(p) при m=80, n=80

7.4. Дополнительное тестирование S-схемы на больших матрицах

S-схема, в отличие от B-схемы, является применимой и для больших матриц при $m < n$. Тестирование проводилось на матрицах размера $m \times n,\ m \in \{30, 40\},\ n \in \{100,\ 150,\ 200\}$. В S-схеме значение параметра r полагалось равным 10.

В таблице 11 приведены время работы $T(p)$ параллельного алгоритма дуализации, основанного на S-схеме. На рисунках 17 – 20 приведены графики $S(p)$ и $E(p)$ для рассматриваемых конфигураций. На рисунках 21 и 22 приведены столбчатые диаграммы для достигнутых уровней загрузки $\sigma_k(p),\ k \in \{1, \dots, p\}$, при $p = 32$.

Как и в разделе 7.3, S-схема демонстрирует практически линейное ускорение при $p \leq 16$. В случае $m = 30$ максимальное ускорение достигается при $p = 32$. На рис. 22 для матрицы 40×150 существует явный выброс при $k = 9$, что говорит о недостаточном качестве оценки $f_r^*(j)$ для этой конфигурации.

Таблица 14. Время работы T(p) для S-схемы

$m \times n \setminus p$	1	2	4	8	16	32	64	128
30×100	3.95	2.03	1.05	0.59	0.37	0.32	0.32	0.32
30×150	39.01	20.00	10.44	5.21	3.46	2.32	2.33	2.32
30×200	231.58	116.91	61.53	32.27	18.85	13.84	13.84	13.84
40×100	11.52	5.83	3.05	1.53	0.96	0.95	0.95	0.95
40×150	133.15	67.12	34.85	19.05	10.90	9.44	9.43	9.43
40×200	654.73	328.66	177.46	90.59	61.86	40.42	36.84	36.89

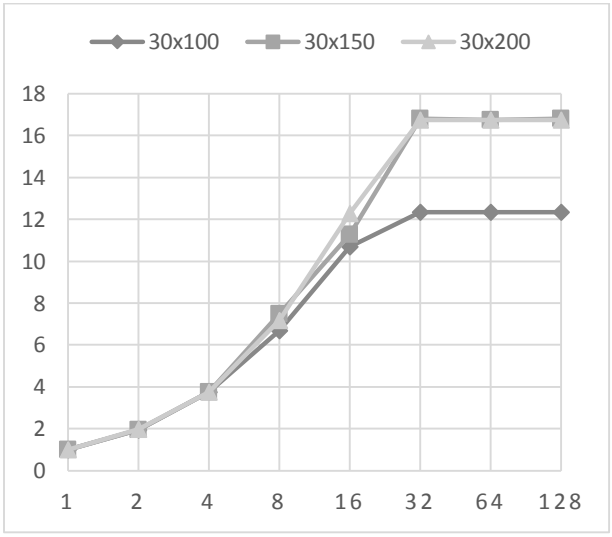


Рис. 17. Графики S(p) для S-схемы при m=30

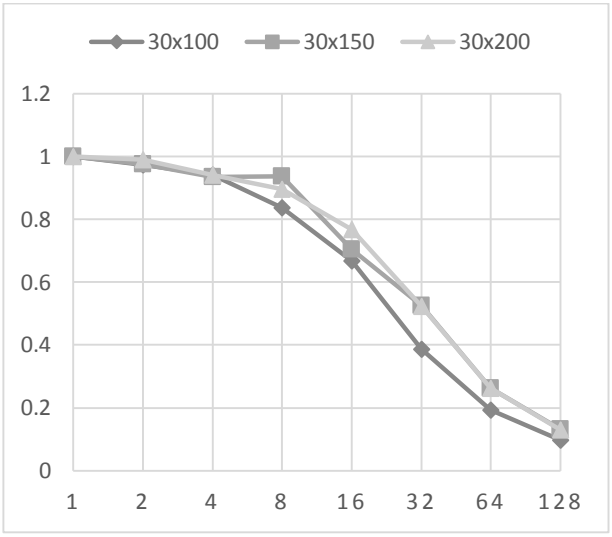


Рис. 18. Графики E(p) для S-схемы при m=30

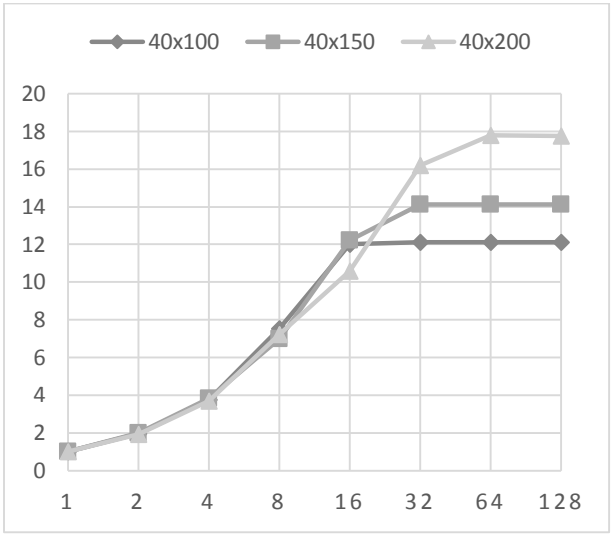


Рис. 19. Графики S(p) для S-схемы при m=40

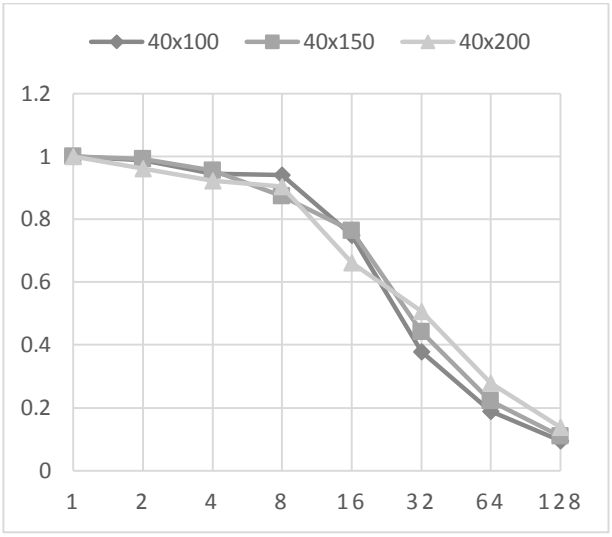


Рис. 20. Графики E(p) для S-схемы при m=40

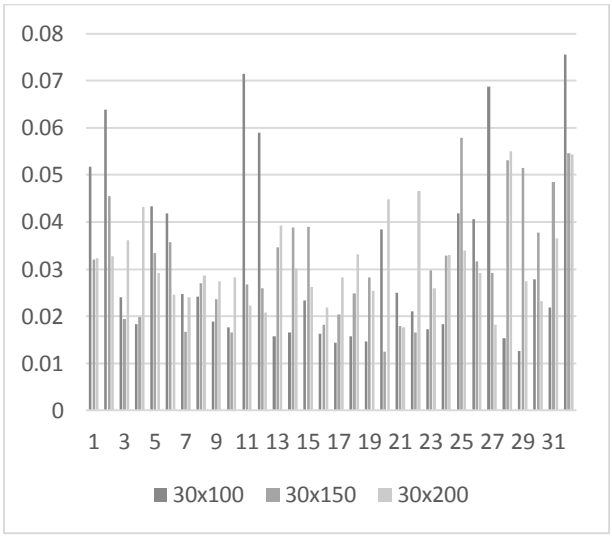


Рис. 21. График $\sigma_k(32)$ при m=30

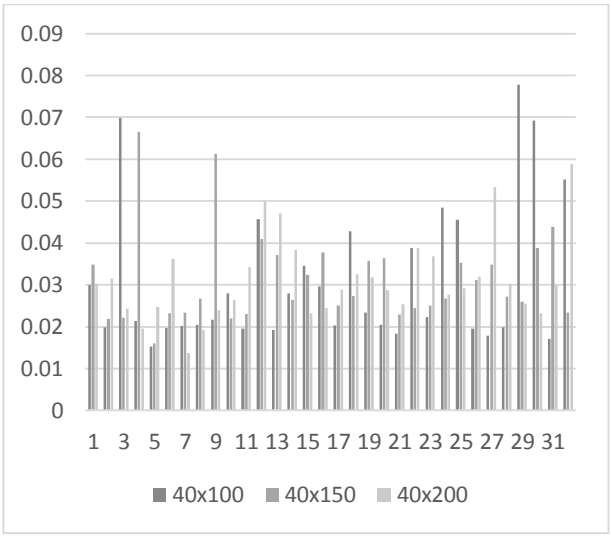


Рис. 22. График $\sigma_k(32)$ при m=40

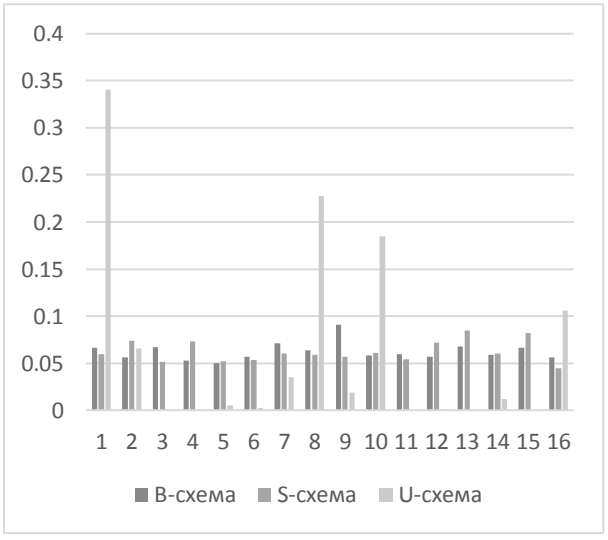


Рис. 23. График $\sigma_k(16)$ при m=80, n=80

8. Заключение

В данной работе получены следующие результаты:

- Предложен новый подход к построению параллельных алгоритмов для дискретных перечислительных задач, проиллюстрированный на примере одной из центральных задач дискретной математики – дуализации;
- В рамках указанного подхода предложены два способа формирования подзадач и распределения их между вычислительными узлами, основанные на статистических оценках объёмов этих подзадач.
- Проведена статистическая обработка экспериментов с целью определения вида распределения случайных величин, определяющих объёмы подзадач. Выявлены условия, при которых обе схемы распараллеливания демонстрируют ускорение, близкое к максимальному, и достаточно равномерную загрузку процессоров;
- Разработан новый асимптотически оптимальный алгоритм дуализации, который значительно превосходит по скорости счета другие существующие алгоритмы дуализации.

9. Список литературы

[1] T. Eiter и G. Gottlob, «Identifying the Minimal Transversals of a Hypergraph and Related Problems,» *SIAM Journal on Computing,* т. 24, № 6, p. 1278–1304, 1991.

[2] Е. Дюкова, «Об асимптотически оптимальном алгоритме построения тупиковых тестов,» *ДАН СССР,* т. 223, № 4, pp. 527 - 530, 1977.

[3] T. Eiter, G. Gottlob и K. Makino, «New results on monotone dualization and generating hypergraph transversals,» *SIAM Journal on Computing,* т. 32, № 2, pp. 514 - 537, 2003.

[4] D. Johnson, M. Yannakis и C. Papadimitriou, «On Generating All Maximum Independent Sets,» *Information Processing Letters,* т. 27, pp. 119 - 223, 1997.

[5] M. Fredman и L. Khachiyan, «On the Complexity of Dualization of Monotone Disjunctive Normal Forms,» *Journal of Algorithms,* т. 21, № 3, pp. 618 - 628, 1996.

[6] L. Khachiyan, E. Boros, K. Elbassioni и V. Gurvich, «An Effcient Implementation of a Quasipolynomial Algorithm for Generating Hypergraph Transversals and its Application in Joint Generation,» *Discrete Applied Mathematics,* т. 154, № 16, pp. 2350 - 2372, 2006.

[7] Е. В. Дюкова, «Асимптотически оптимальные тестовые алгоритмы в задачах распознавания,» *Пробл. кибернетики,* № 39, pp. 165-199, 1982.

[8] Е. Дюкова, «О сложности реализации дискретных (логических) процедур распознавания,» *Журнал вычислительной математики и математической физики,* т. 44, № 3, pp. 551 - 561, 2004.

[9] Е. В. Дюкова и Ю. И. Журавлёв, «Дискретный анализ признаковых описаний в задачах распознавания большой размерности,» *Ж. вычисл. матем. и матем. физ.,* т. 40, № 8, pp. 1264-1278, 2000.

[10] E. V. Djukova и Y. I. Zhuravlev, «Discrete methods of information analysis in recognition and algorithm synthesis,» *Pattern Recognition and Image Analisys,* т. 7, № 2, pp. 192-207, 1997.

[11] K. Murakami и T. Uno, «Efficient Algorithms for Dualizing Large-Scale Hypergraphs,» *Discrete Applied Mathematics,* т. 170, pp. 83 - 94 , 2014.

[12] Е. Дюкова и П. Прокофьев, «Построение и исследование новых асимптотически оптимальных алгоритмов дуализации,» *Машинное обучение и анализ данных,* т. 1, № 8, pp. 1048 - 1067, 2014.

[13] Е. Дюкова и А. Инякин, «Асимптотически оптимальное построение тупиковых покрытий целочисленной матрицы,» *Математические вопросы кибернетики,* т. 17,

pp. 235 - 246, 2008.

[14] L. Kachiyan, E. Boros, K. Elbassioni и V. Gurvich, «A New Algorithm for the Hypergraph Transversal Problem,» *Lecture Notes in Computer Science,* т. 3595, pp. 767 - 776 , 2005.

[15] L. Khachiyan, E. Boros, V. Gurvich и K. Elbassioni, «Computing many maximal independent sets for hypergraphs in parallel,» *Parallel Processing Letters,* т. 17, № 2, pp. 141 - 152, 2007.

[16] Е. Дюкова, А. Никифоров и П. Прокофьев, «Статистически эффективная схема распараллеливания алгоритмов дуализации,» *Машинное обучение и анализ данных,* т. 1, № 7, pp. 846 - 853, 2014.