Информационные процессы, Том 11, № 4, 2011, стр. 476–499.

МАТЕМАТИЧЕСКИЕ МОДЕЛИ, ВЫЧИСЛИТЕЛЬНЫЕ МЕТОДЫ

Кодирование состояний в методе матрицы переноса для подсчета гамильтоновых циклов на прямоугольных решетках, цилиндрах и торах

А.М.Караваев

karavaev@flowproblem.ru, Петрозаводский государственный университет, Петрозаводск, Россия

Поступила в редколлегию 15.12.2011

Аннотация—Предлагается эффективная вычислительная реализация метода матрицы переноса в задаче подсчета гамильтоновых циклов на семействах прямоугольных решеток, цилиндров и торов. Впервые получены рекуррентные соотношения, описывающие количество гамильтоновых циклов на некоторых семействах торов.

1. ВВЕДЕНИЕ

В работе рассматриваются три семейства графов: прямоугольные решетки $P_m \times P_n$, цилиндры $C_m \times P_n$ и торы $C_m \times C_n$. Любая решетка порождается прямым произведением двух простых цепей P_m и P_n, первая из которых имеет m вершин, а вторая — n. Будем называть число m шириной решетки (протяженность слева направо), а n — ее длиной (протяженность снизу вверх). Символом $C_m \times P_n$ обозначается цилиндр, представляющий собой прямое произведение цикла C_m из m вершин (периметр цилиндра) и цепи P_n из n вершин (высота, или образующая цилиндра). Тор $C_m \times C_n$ является прямым произведением двух циклов C_m и C_n.

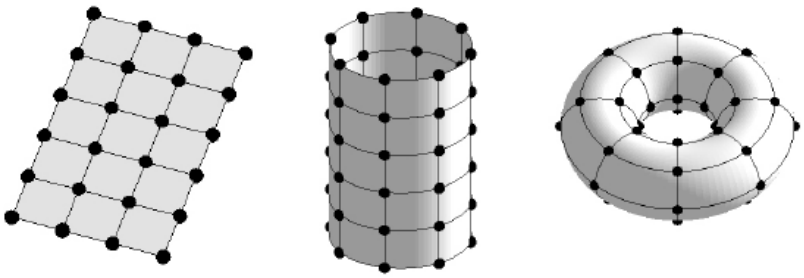


Рис 1. Примеры сеточных графов.
Слева направо: решетка $P_4 \times P_6$, цилиндр $C_8 \times P_6$ и тор $C_6 \times C_8$

Гамильтоновым циклом в графе называется цикл, проходящий через каждую вершину графа в точности один раз. Поскольку другие циклы в работе не обсуждаются, то гамильтонов цикл будем для краткости называть просто циклом.

Наиболее эффективным способом подсчета циклов в сеточных графах является использование метода матрицы переноса. Его основное достоинство состоит в том, что он позволяет подсчитать количество циклов в графе без их фактического построения, что особенно важно для задач, принадлежащих классу сложности $\sharp\mathtt{P}$ [1]. Другой важной особенностью этого метода оказывается то, что числовая последовательность, полученная при фиксированном значении одного из параметров семейства, который в дальнейшем будем обозначать через m, удовлетворяет линейному рекуррентному соотношению с постоянными коэффициентами. Решения найденных соотношений дают аналитическое описание числа циклов для всего семейства.

Логическая простота метода сводит проблему подсчета циклов к вычислению степени матрицы переноса, однако следствием ее вычислительной сложности становится экспоненциальный относительно m рост порядка матрицы. Таким образом, залогом успешного применения метода матрицы переноса является наличие эффективной схемы кодирования состояний и максимально возможное сжатие матрицы путем удаления недостижимых и отождествления идентичных состояний.

Основной целью работы стало обобщение и оптимизация описанных ранее в [2] приемов на все три семейства сеточных графов. Благодаря предложенной схеме кодирования удалось вывести явные выражения для числа циклов на рассматриваемых семействах графов в более широком диапазоне изменения параметра m. В случае торов такие выражения получены впервые [3].

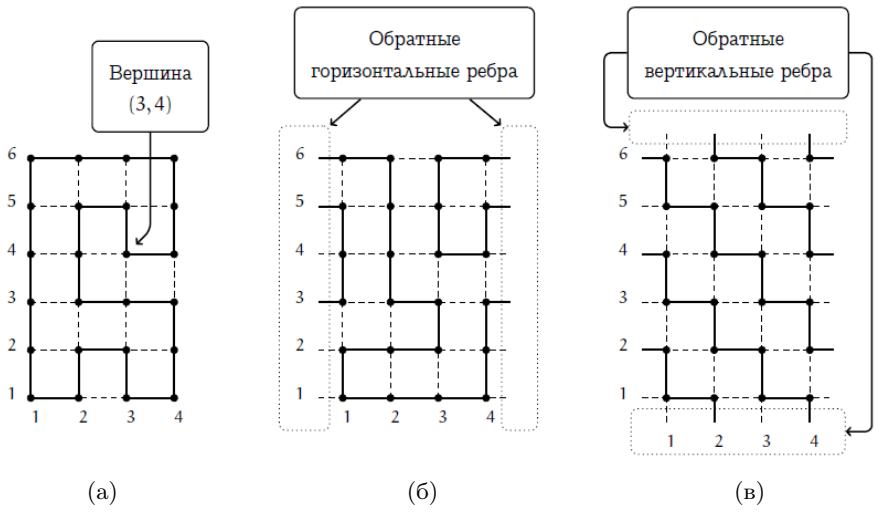
2. ТЕРМИНОЛОГИЯ И СХЕМА АЛГОРИТМА

2.1. Вспомогательная терминология

Рассматриваемые в работе сеточные графы являются графами декартовых произведений, поэтому структуру всех трех семейств можно с единых позиций описать в терминах уровней. Например, решетку $P_m \times P_n$ удобно представить в виде n копий графа P_m, связанных друг с другом по определенному правилу. Каждую из этих копий будем называть *уровнем* или *слоем*.

Перенумеруем все имеющиеся уровни числами от 1 до n. Если теперь последовательно пометить вершины каждой отдельно взятой цепи P_m числами от 1 до m, присвоив значения 1 и m концевым вершинам, то все вершины решетки могут быть однозначно помечены парой целых чисел (i, k). Первое из чисел указанной пары будет соответствовать порядковому номеру вершины в одной из копий цепи P_m, второе — номеру уровня. Ребра решетки, принадлежащие одной и той же копии цепи P_m будут называться *горизонтальными*. Два различных слоя соединяются между собой *вертикальными* ребрами, если их номера отличаются на 1. Вертикальные ребра инцидентны вершинам с одинаковыми номерами внутри слоя.

Решетку будем изображать на плоскости, нумеруя ее слои снизу вверх и располагая их друг над другом так, чтобы горизонтальные ребра были изображены горизонтально, а вертикальные — вертикально, так, как указано на рис. 2(а).



(а) (б) (в)

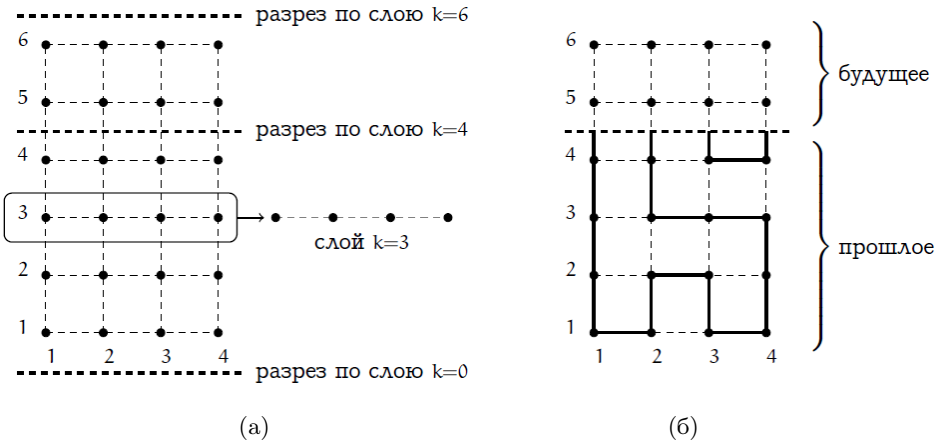
Рис 2. Развертка сеточных графов на плоскости вместе гамильтоновыми циклами. Слева направо: решетка $P_4 \times P_6$, цилиндр $C_4 \times P_6$ и тор $C_4 \times C_6$

Цилиндр может быть получен из решетки путем наложения циклических граничных условий на ее левую и правую границы. Поскольку любая цепь P_m трансформируется в цикл C_m, путем добавления ребра, связывающего концевые вершины цепи, цилиндр допускает представление в виде n копий цикла C_m, связанных друг с другом по тому же принципу, как и в случае решеток. При использовании изложенной выше схемы разметки вершин, добавленные ребра будут для всех слоев k инцидентны вершинам (m, k) и $(1, k)$. Будем называть такие ребра *обратными горизонтальными ребрами*. Именно они и обеспечивают циклические граничные условия. (рис. 2(б)).

Тор может быть представлен в виде n копий цикла C_m, соединенных по той же самой схеме, что и в случае цилиндра, но с дополнительным условием; слои с номерами 1 и n соединяются друг с другом *обратными вертикальными ребрами*. Эти ребра инцидентны вершинам с одинаковыми номерами внутри слоя и также как на развертке цилиндра изображаются короткими отрезками (рис. 2(в)).

Разрез сеточного графа по слою k — это разбиение вершин графа на два подмножества так, чтобы вершины слоев с номерами $j \leq k$ оказались в одном подмножестве (будем называть его нижним), а вершины слоев с номерами $j > k$ — в другом, которое будем называть верхним. Принятая нами разметка вершин позволяет изобразить разрез пунктирной линией, как показано на на рис. 3(а). Визуально линия разреза всегда будет изображаться выше того слоя, по которому производится разрез, чтобы не сливаться с входящими в него вершинами и горизонтальными ребрами.

Для единообразного описания алгоритма полезно ввести разрезы по слоям $k = 0$ и $k = n$. Первый из этих разрезов (по несуществующему нулевому слою) отделяет пустое пространство под графом от самого графа, а второй отделяет граф от пустого пространства над ним.



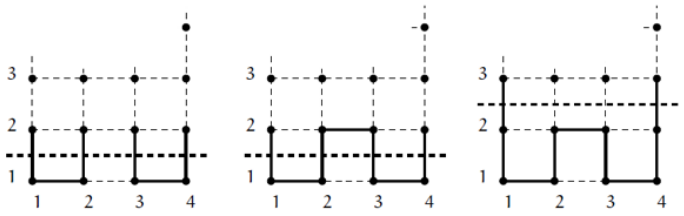
(а) (б)

Рис 3. Слои (уровни) и разрезы в графе $P_4 \times P_6$:
(а) Примеры слоев и разрезов; (б) Построенная в прошлом часть «цикла»

Разрез делит множество вершин графа на два подмножества: верхнее и нижнее. Подграф сеточного графа, индуцированный нижним подмножеством вершин, будем называть *прошлым*. Аналогично, подграф, индуцированный верхним подмножеством вершин — *будущим* (рис. 3(б)). Введение этих понятий призвано формализовать описание алгоритма построения цикла в любом из графов рассматриваемых семейств.

2.2. Общая схема алгоритма

Построение цикла представляет собой итерационный процесс, на каждой итерации которого линия разреза перемещается от одного уровня k к следующему, причем вначале $k = 0$. Первая итерация начинается с выбора некоторых горизонтальных ребер на уровне 1, по которым будет проходить цикл. Ребра выбираются произвольно, но так, чтобы они были инцидентны всем вершинам 1-го слоя, иначе условие гамильтоновости окажется нарушенным. Множество выбранных горизонтальных ребер единственным образом определяет набор вертикальных ребер, соединяющих первый слой со вторым. После фиксации вертикальных ребер линия разреза перемещается к слою $k = 1$ (рис. 4(а)).



(а) (б) (в)

Рис 4. Этапы построения цикла: (а) начальная итерация; (б) выбор горизонтальных ребер; (в) фиксация вертикальных ребер и перемещение линии разреза на следующий уровень

На последующих итерациях (рис. 4(б-в)) снова выбираются горизонтальные ребра, однако теперь их выбор ограничен структурой цикла и поэтому менее произвольный. Действительно, проходящая через прошлое часть цикла представляет собой множество непересекающихся простых цепей, концы которых находятся на линии разреза, а сами цепи проходят по всем вершинам прошлого (рис. 3(б)). В связи с этим, следует учитывать еще два правила, невыполнение которых будет нарушать гамильтоновость цикла. Первое из правил запрещает инцидентность более двух ребер строящегося цикла одной вершине, поскольку цикл является регулярным остовным подграфом степени 2.

Второе правило требует чтобы ни одна из построенных в прошлом цепей не замыкалась в цикл, поскольку замыкание должно происходить лишь на последней итерации. Если не соблюдать это правило, то вместо гамильтонового цикла будет построен некоторый 2-фактор графа. Неудачный выбор горизонтальных ребер в прошлом может привести к такой ситуации, когда замыкание на последней итерации окажется невозможным.

На каждой итерации построения цикла можно выделить три основных этапа: выбор горизонтальных ребер, фиксация вертикальных ребер единственно возможным способом и перемещение линии разреза на один уровень вверх. Совокупность этих трех действий, выполняемых в указанном порядке, будем называть *шагом в будущее*. Уровень, на котором находится линия разреза, называется *текущим*.

2.3. Состояния и переходы

Из изложенного выше следует, что структура цикла накладывает ряд требований на то, какие горизонтальные ребра могут быть выбраны в каждом слое. В работах, посвященных методу матрицы переноса, эти требования принято описывать в терминах «состояние» и «переход». При подсчете циклов в сеточных графах соответствующие термины естественным образом связываются с понятиями прошлого и будущего. Под состоянием принято понимать способ расположения цепей в прошлом друг относительно друга и текущего уровня в некоторый фиксированный момент работы алгоритма.

Основной проблемой при определении понятия состояния является формализация того набора требований, которые должны быть выполнены для успешного построения цикла. Для каждого из рассматриваемых семейств набор этих требований несколько отличается и соответствующие уточнения будут сделаны в разделах, посвященных кодированию состояний. В то же время, структура всех сеточных графов имеет ряд общих свойств и эта общность должна найти свое отражение в используемой схеме кодирования. Характерные для всех трех семейств подходы к кодированию проще всего проиллюстрировать на примере решеток.

Рассмотрим изображенную на рис. 5 слева решетку $P_6 \times P_8$, цикл в ней и разрез по слою $k = 5$. Прошлое выделено пунктирной рамкой. Расположенная в прошлом часть «цикла» состоит из двух цепей. В будущем изображен один из вариантов соединения этих цепей.

Для того, чтобы сделать один шаг в будущее — сдвинуть линию разреза и достроить «цикл» до слоя $k = 6$, — достаточно знать информацию о взаимном расположении цепей, а точнее — о взаимном расположении концевых вершин этих цепей на текущем уровне. Требуемую информацию можно представить в виде одного слоя графа, на котором концы цепей соединены дугообразными линиями, как показано на рис. 5 справа. Каждой дуге ставится в соответствие одна цепь из прошлого.

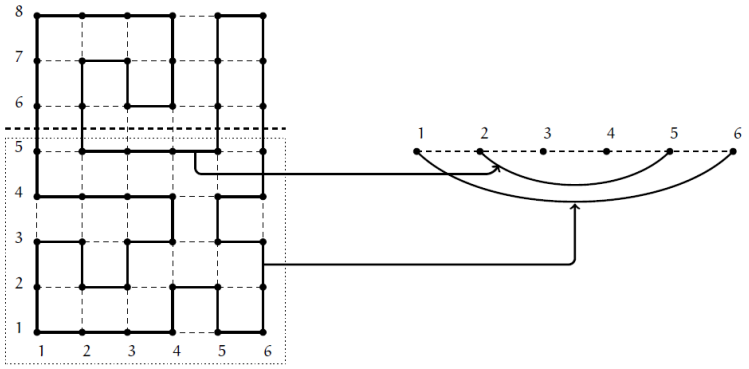


Рис 5. Допустимое состояние, возникающее при построении цикла на уровне $k = 5$

Множество дуг, возможно пустое, закодированное особым образом, будем называть *состоянием* на решетке. Способ такого представления состояния не зависит от номера слоя. Каждую дугу можно обозначить парой чисел (p, q), в которой p и q — номера вершин слоя, соединенные этой дугой. Будем предполагать, что всегда выполняется условие $p < q$.

Сформулированные в предыдущем разделе правила накладывают ограничения на взаимное расположение дуг в каждом состоянии. Если состояние содержит больше одной дуги, то упорядочив каждую пару дуг (p_1, q_1) и (p_2, q_2) так, чтобы выполнялось неравенство $p_1 < p_2$, потребуем выполнения условия $p_2 > q_1 \vee q_2 < q_1$. С учетом принятого способа изображения, это условие соответствует тому, что либо концы одной дуги находятся «правее» концов другой, либо между ними. Будем считать, что для состояний, содержащих только одну дугу или вообще без дуг, данные условия всегда выполняются. Состояние, удовлетворяющее сформулированным выше требованиям, назовем *возможным состоянием*, а всю их совокупность — *множеством возможных состояний*.

Особый случай состояния с пустым множеством дуг соответствует разрезам по слоям $k = 0$ и $k = n$. Для удобства описания алгоритма полезно различать эти ситуации. Состояние, соответствующее значению $k = 0$ будем называть *начальным*, а случаю $k = n$ — *финальным*.

При последовательном построении цикла снизу вверх могут быть получены не все возможные состояния, что обусловлено как свойствами решетки, так и самого цикла. Данное

обстоятельство позволяет разбить множество всех возможных состояний на множество *допустимых* (*достижимых*) состояний и множество *недопустимых* (*недостижимых*) состояний. Начальное и финальное состояния по определению будем считать допустимыми. Количество допустимых состояний будет обозначаться через M. Для удобства ссылок на конкретное состояние, перенумеруем их произвольным образом числами от 1 до M так, чтобы начальное состояние имело номер 1, а финальное — M.

Следует отметить, что сама форма цикла полностью реконструируется последовательностью допустимых состояний, которые были получены в процессе его построения. Особенностью описанного в настоящей работе подхода является такая схема вычислений, при которой недопустимые состояния вообще не рассматриваются.

Переходом из допустимого состояния s в допустимое состояние t называется возможность выполнения такого шага в будущее, при котором из состояния s на одном уровне получается состояние t на следующем уровне сеточного графа. В этом случае будем говорить, что из состояния s можно перейти в состояние t. На рис. 6 показаны переходы, возникающие при построении цикла на решетке $P_6 \times P_8$ (см. рис. 5).

(а) (б)

Рис 6. Переходы между состояниями, показанными на рис. 5:
(а) от слоя $k = 2$ к слою $k = 3$; (б) от слоя $k = 5$ к слою $k = 6$

В терминах теории графов множество допустимых состояний можно ассоциировать с множеством вершин некоторого псевдографа, в котором переходы являются его ориентированными ребрами. Это позволяет свести задачу подсчета циклов к более простой задаче — подсчету количества маршрутов между двумя фиксированными вершинами псевдографа.

3. МЕТОД МАТРИЦЫ ПЕРЕНОСА И ИСТОРИЯ ЕГО ПРИМЕНЕНИЯ

3.1. Метод матрицы переноса

Каждый маршрут длиной n из начальной вершины псевдографа в финальную однозначно определяет некоторый цикл на решетке $P_m \times P_n$. Количество таких маршрутов может быть найдено, если опираться на интерпретацию элементов степеней матрицы смежности этого псевдографа. В силу особого статуса этой матрицы будем называть ее *матрицей переноса* и обозначать через T. Тогда с учетом принятого выше правила нумерации состояний, искомое количество маршрутов равно $(T^n)_{1,M}$.

В качестве примера рассмотрим решетку $P_3 \times P_6$. На рис. 7 слева показан цикл в этом графе, который проходит через все возможные состояния. Для $m = 3$ множества возможных и допустимых состояний совпадают. На этом же рисунке справа изображен псевдограф переходов между состояниями. В вершинах псевдографа, соответствующих финальному и начальному состояниям, изображены цепи P_3. Для визуального отличия состояний в первом случае ребра имеют вид сплошных линий, а во втором — пунктирных.

Количество способов перейти из состояния 1 в состояние $M = 5$ за n шагов в таком псевдографе равно количеству циклов на решетке $P_3 \times P_n$.

Посчитать требуемую величину можно, построив матрицу переноса T и возведя ее в степень n. Когда $m = 3$, а $n = 6$, матрица T и ее степень имеют вид

$$T = \begin{pmatrix} 0\,1\,0\,0\,0 \\ 0\,0\,1\,1\,1 \\ 0\,1\,0\,0\,0 \\ 0\,1\,0\,0\,0 \\ 0\,0\,0\,0\,0 \end{pmatrix}, \qquad T^6 = \begin{pmatrix} 0\,0\,4\,4\,4 \\ 0\,8\,0\,0\,0 \\ 0\,0\,4\,4\,4 \\ 0\,0\,4\,4\,4 \\ 0\,0\,0\,0\,0 \end{pmatrix}.$$

Величина $(T^6)_{1,5} = 4$ согласуется с результатами [4], где было показано, что число циклов на решетке $P_3 \times P_n$ равно $2^{n/2-1}$ при четном n и равно нулю при нечетном значении n.

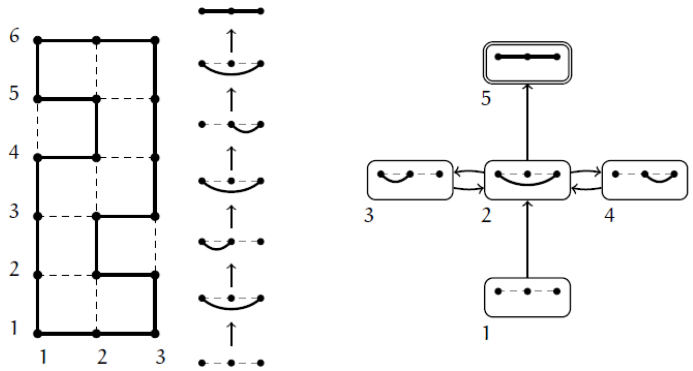


Рис 7. Слева: цикл на решетке $P_3 \times P_6$, проходящий через все возможные состояния. Справа: псевдограф переходов между состояниями при $m = 3$

На практике такой способ решения задачи применим лишь для небольших m, когда матрица T целиком умещается в оперативной памяти компьютера. Порядок матрицы переноса для фиксированного m можно оценить сверху числами Моцкина [5, 6, 7]:

$$\mathcal{M}_m = \sum_{k=0}^{\lfloor m/2 \rfloor} \frac{1}{k+1} \binom{2k}{k} \binom{m}{2k} \sim \sqrt{\frac{3}{4\pi m^3}} \cdot 3^m.$$

Из этой оценки следует, что при увеличении m на единицу затраты памяти на хранение исходной матрицы возрастают почти в 9 раз.

Фактически ситуация гораздо хуже, поскольку при возведении в степень элементы матрицы экспоненциально растут. Если, как в [8], при подсчете циклов на решетках $P_{11} \times P_n$ использовать все 5 797 возможных состояний, то при $n = 100$ для точного представления в памяти компьютера отдельно взятого элемента матрицы T^n потребуется более 64 байт. В итоге общие затраты памяти только на хранение степени матрицы переноса превысят 2 ГиБ.

Применение серии приемов из раздела 4.3 «Оптимизация» (с. 490) хоть и позволяет сократить количество допустимых состояний, но не позволяет избавиться от экспоненциального роста этой величины. Так, при выводе рекуррентных соотношений для $m = 11$ (см. раздел 6) необходимо определять число циклов на решетках $P_{11} \times P_n$ при значениях параметра $n \geq 1\,000$. Однако даже с учетом оптимизаций, уменьшающих число допустимых состояний до 2 355, элементы матрицы $T^{1\,000}$ будут иметь длину более 3 000 десятичных знаков, что потребует почти 7 ГиБ памяти. Для решетки шириной $m = 22$ порядок матрицы T с учетом оптимизаций равен 70 652 188 (табл. 1, с. 490).

Все сказанное выше свидетельствует о необходимости разработки таких алгоритмов, которые бы позволили вычислить требуемый элемент любой фиксированной степени матрицы переноса без явного хранения в памяти всей матрицы.

Несмотря на то, что метод матрицы переноса является наиболее эффективным способом подсчета циклов в сеточных графах, как показывают сделанные выше оценки, он является чрезвычайно ресурсоемким и не может быть использован при повседневных расчетах. Подавляющая часть результатов, полученных этим методом к настоящему времени, приводится в виде табличных данных, что существенно затрудняет или даже делает невозможным их анализ, особенно в тех случаях, когда требуется сопоставление каких-либо величин, отсутствующих в таблицах. По указанным причинам было бы крайне желательно иметь аналитическое представление для полученных результатов, по крайней мере, в пределах некоторого фиксированного семейства графов.

К счастью, природа самого метода матрицы переноса такова, что полученные с его помощью числовые последовательности $a_n = (T^n)_{1,M}$ будут удовлетворять линейным рекуррентным соотношениям с постоянными коэффициентами. Наличие такой возможности, являющейся следствием теоремы Кэли — Гамильтона, позволяет, в частности, оценить сверху порядок соотношения величиной M.

Использование рекуррентных соотношений может представлять интерес по ряду причин. Во-первых, с их помощью удобно систематизировать многочисленные, разбросанные по различным источникам, данные. Во-вторых, наличие средств для работы с рекуррентными соотношениями в развитых системах компьютерной алгебры позволяет эффективно вычислять нужный элемент последовательности практически за линейное относительно его номера время. В-третьих, для приближенных вычислений на основе корней характеристического полинома можно без труда построить асимптотику последовательности.

По сути дела, вывод рекуррентного соотношения позволяет получить исчерпывающую информацию о количестве циклов в данном семействе графов и, как следствие, считать задачу полностью решенной для данного значения m.

В рассмотренном выше случае $m = 3$ характеристический полином матрицы T равен $|T - \lambda I| = \lambda^5 - 2\lambda^3$. Из этого следует, что последовательность (a_n) удовлетворяет соотношению $a_n = 2a_{n-2}$ при $n \geq 5$, начальными данными для которого будут значения $a_1 = 0$, $a_2 = 1$, $a_3 = 0$, $a_4 = 2$. Из начальных данных видно, что соотношение справедливо для $n \geq 3$.

Следует отметить, что алгоритмы для вывода рекуррентных соотношений можно условно разделить на две группы. Методы, составляющие первую группу, позволяют выводить соотношение непосредственно из матрицы T, путем вычисления ее характеристического полинома, как это было показано в случае $m = 3$. Однако применение таких методов едва ли возможно для матриц достаточно большого размера, которые не умещаются в память компьютера.

Идеи, лежащие в основе другой группы методов, позволяют построить рекуррентное соотношение по некоторому конечному отрезку последовательности (a_n). Длина начального отрезка никогда не превосходит $2M$. Методы второй группы значительно эффективнее методов первой, однако детальное их изложение выходит за рамки работы.

3.2. История применения метода

Первая практичная схема кодирования состояний была предложена на рубеже 80-х годов прошлого столетия при изучении свойств полимерных систем [5,9,10]. В этих работах молекула полимера моделировалась самонепересекающимся блужданием на решетке, что позволяло применить для решения задач метод матрицы переноса. Систематическое исследование числа циклов на решетках и цилиндрах с применением схемы кодирования [5,9] было проведено в [8].

Следует отметить, что в работах физической направленности задача точного подсчета числа циклов, как правило, не ставится. Дело в том, что при соответствующем выборе единиц измерения и переходе к безразмерным величинам непосредственный физический смысл имеет

логарифм числа циклов, тесно связанный с энтропией, приходящейся на один узел решетки. Данную величину принято называть константой связности решетки.

Если при расчетах ограничиться лишь оценкой константы связности, то достаточно вычислить только главное собственное значение матрицы переноса. Подобный подход не меняет вычислительную сложность задачи, однако может существенно упростить реализацию, поскольку не предполагает поддержку арифметики целых чисел произвольной точности. С учетом сказанного неудивительно, что при использовании встроенной машинной арифметики в [8] представлены точные значения числа циклов только для таких решеток $P_m \times P_n$, для которых $m, n \leq 10$.

Наиболее серьезным ограничением работы [8] является использование множества всех возможных состояний при построении матрицы T. В результате ширина решеток, для которых удавалось разместить матрицу переноса в памяти компьютера не превышала 11.

В [7] для удаления из матрицы переноса заведомо недостижимых состояний было введено понятие четности состояния. Опираясь на соображения четности, авторам удалось существенно понизить порядок матрицы в случае четных значений m. Для этой же цели в [6] применялись алгоритмы минимизации конечного автомата. В рамках более формального подхода наблюдалось ощутимое уменьшение порядка матрицы для всех, а не только для четных m. Однако, несмотря на то, что в цитированных работах были успешно подсчитаны циклы в некоторых сеточных графах и получено несколько новых рекуррентных соотношений, их авторам не удалось расширить диапазон значений m, исследованных в [8].

Метод матрицы переноса допускает не только численную, но и символьную реализацию. Именно такая реализация использовалась в [11]. Для подсчета подграфов определенного вида в графах декартовых произведений $G \times P_n$ автор строил матрицу переноса и применял различные подходы для вывода рекуррентных соотношений относительно параметра n. Такие соотношения для количества циклов на цилиндрах $C_m \times P_n$ при $m \leq 5$ следуют из результатов [11] в качестве частных случаев.

Границы применимости символьной реализации метода до конца не изучены. Дело в том, что при выводе рекуррентного соотношения требуется либо построить характеристический полином матрицы, что весьма затруднительно для матриц порядком несколько тысяч, либо вычислить при фиксированном m достаточно много начальных членов последовательности, элементы которой растут экспоненциально. Коэффициенты рекуррентного соотношения могут быть затем найдены из решения системы линейных уравнений, матрица и свободный член которой составлены из элементов последовательности. При использовании второго подхода необходимо задействовать арифметику с произвольной точностью.

В любом случае, символьная реализация метода исключает возможность традиционного «ручного» вывода соотношений, за исключением наиболее простых случаев, подобных [12,13].

Лучший среди известных алгоритмов с хранением матрицы переноса был предложен в [14]. Это первая работа, в которой описана схема кодирования состояний на цилиндрах, оказавшаяся значительно более эффективной, чем в [8,11]. Используя разреженную структуру матрицы переноса, автору удалось резко сократить объем вычислений, что позволило провести расчеты для цилиндров $C_m \times P_n$ при $m \leqslant 14$.

Существенное сжатие матрицы переноса становится возможным если не различать такие допустимые состояния, количество переходов в которые одинаково. Для этой цели имеет смысл воспользоваться соображениями симметрии [15]. Действительно, поскольку цепь P_m симметрична относительно своей середины, то для решеток можно примерно в два раза сократить количество допустимых состояний для четных m по сравнению с [7]. Совмещение простого цикла C_m с самим собой при повороте на угол $2\pi/m$ позволяет добиться m кратного сокраще-

ния порядка матрицы в случае цилиндров. В алгоритмах подсчета циклов такая возможность ранее не использовалась.

Радикальным средством экономии памяти могла бы стать такая модификация метода, в которой матрица переноса явно не фигурировала. Сравнительно недавно такая возможность была продемонстрирована в [16]. Несмотря на то, что использовавшаяся в [16] схема кодирования состояний была далека от оптимальной, автору удалось вычислить количество циклов на квадратной решетке $P_{20} \times P_{20}$.

Задача о подсчете циклов на семействах торов значительно сложнее, чем для двух других семейств. Единственные точные результаты приводятся в [17], автор которой определил количество циклов на торах $C_m \times C_n$ с нечетным числом узлов, при условии $mn \leq 39$. Исходя из этих ограничений, можно предположить, что при расчетах использовался метод полного перебора.

Все особенности метода, описанные в данном разделе, легли в основу параллельного алгоритма, основанного на принципах динамического программирования [2]. Благодаря использованию описанной в работе схемы кодирования, набор графов, допускающих точный подсчет, был существенно расширен вплоть до $P_{22} \times P_{100}$ и $C_{22} \times P_{100}$. Для меньших значений m удалось вычислить последовательности, длина которых оказалась достаточной для того, чтобы реконструировать соответствующие им рекуррентные соотношения. С количественными данными можно ознакомиться на сайте [18] в разделе о гамильтоновых циклах.

4. КОДИРОВАНИЕ СОСТОЯНИЙ НА РЕШЕТКАХ $P_m \times P_n$ И ЦИЛИНДРАХ $C_m \times P_n$

Каждое из возможных состояний на решетке представляет собой множество (быть может, пустое) дуг, соединяющих вершины цепи P_m так, как указано в разделе 2.3. В настоящем разделе предлагается способ кодирования допустимых состояний с помощью правильных скобочных последовательностей, разреженных нулями, а также обоснование тому, что такой же способ кодирования подходит для случая цилиндров. В дальнейшем изложении, если не оговорено иное, под словом «состояние» будем понимать именно допустимое состояние, поскольку наш алгоритм подсчета циклов работает только с состояниями такого вида.

Введем *нулевую* последовательность длиной k следующим образом: $\texttt{0}^k = \texttt{00...0}$ (k нулей). Длина k может равняться нулю, что соответствует пустой последовательности. Пусть $\texttt{0}^*$ — последовательность нулей с произвольной длиной k от 0 до ∞. Правильная скобочная последовательность, разреженная нулями (в дальнейшем, ПСПРН) определяется как последовательность символов «(», «)» и «0», подчиняющаяся контекстно-свободной грамматике

$$\texttt{ПСПРН} \to \texttt{0}^* \mid \texttt{0}^*\texttt{(ПСПРН)ПСПРН}.$$

Длиной последовательности называется количество символов в ней (пустая последовательность имеет длину 0). Среди всех цепочек, порождаемых указанной грамматикой отберем только те из них, которые имеют длину m.

Зафиксируем некоторое состояние s, не являющееся начальным или финальным. Это состояние можно закодировать с помощью ПСПРН. Действительно, Каждой дуге $(p, q) \in s$ поставим в соответствие пару скобок: открывающую «(» и закрывающую «)» — открывающую скобку установим в позицию p, а закрывающую — в позицию q. После этого, тем позициям последовательности, в которых ни одна из скобок не установлена, присваиваем значение «0». В принятом нами способе изображения состояний, при котором вершины цепи P_m изображаются слева направо в соответствии с нумерацией, такая схема кодирования будет соответствовать ПСПРН (рис. 8).

Начальное и финальное состояния будем кодировать последовательностью $\mathtt{0}^m$. Но чтобы различать их, первое будем отмечать чертой снизу, а второе — чертой сверху. Таким образом, начальное состояние имеет вид $\underline{\mathtt{0}^m}$, а финальное — $\overline{\mathtt{0}^m}$.

Рис 8. Примеры соответствия символического изображения состояния правильной скобочной последовательности, разреженной нулями

Способ кодирования состояний с помощью ПСПРН удобен по той причине, что, во-первых, наглядная иллюстрация состояния в виде множества непересекающихся дуг вызывает естественные аналогии с правильной скобочной последовательностью, во-вторых, в программной реализации такая последовательность допускает эффективный способ хранения в виде набора из $2m$ бит. Символ «0» в предложенном способе кодирования выбран для удобства обозначения «отсутствия» дуги, инцидентной соответствующей вершине цепи P_m.

Предложенный способ определения и кодирования состояний является корректным в том смысле, что он позволяет описать любое состояние. Докажем это.

Прошлое является планарным графом, а построенные в прошлом цепи не могут пересекаться по вершинам решетки. Это означает, что если мы зафиксируем некоторое состояние s, содержащее более одной дуги, то любые две дуги (p_1, q_1), $(p_2, q_2) \in s$, для которых $p_1 < p_2$, будут обладать свойством $p_2 > q_1 \vee q_2 < q_1$. В принятом способе кодирования это означает, что либо одна дуга находится правее другой, то есть их взаимное расположение будет таким же, как, например, в ПСПРН ()(), либо одна дуга находится внутри другой, что схематически можно обозначить как (()). Состояние с одной дугой может выглядеть как последовательность (). Нули «0» разрежают последовательность скобок в зависимости от того, к каким вершинам цепи P_m не прикреплена дуга.

Как и для решетки, построение цикла на цилиндре начинается с состояния $\underline{\mathtt{0}^m}$. Если мы покажем, что при переходе из состояния s в какое-либо другое, способ кодирования не меняется, то из этого будет следовать, что все допустимые состояния на цилиндре кодируются точно также, как в случае решетки.

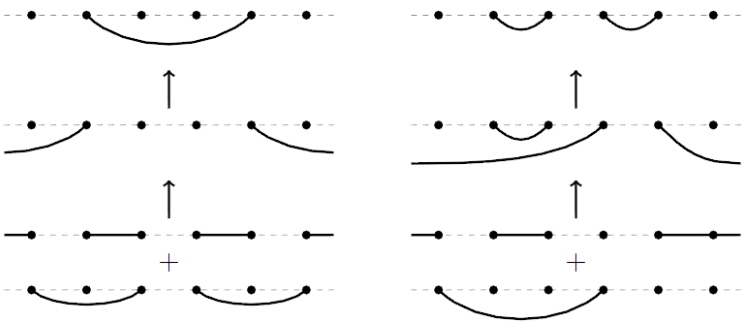
Цилиндр может быть получен из решетки добавлением обратных горизонтальных ребер, соединяющих вершины правой границы с вершинами левой. По аналогии с решеткой, *слоем* (*уровнем*) цилиндра является цикл C_m. Разрез по слою определятся в точности также, как для решетки. Достаточно показать, что произвольное состояние s для цилиндра также может быть закодировано множеством дуг, соединяющих вершины цикла C_m, чтобы для любых двух дуг (если их не меньше двух) (p_1, q_1), $(p_2, q_2) \in s$ для которых $p_1 < p_2$, выполнялось свойство $p_2 > q_1 \vee q_2 < q_1$. Это автоматически гарантирует, что состояния на цилиндре допускают запись в виде ПСПРН.

Описанный в разделе 2.2 алгоритм построения цикла на решетке естественным образом расширяется на случай цилиндра. Действительно, при переходе в будущее все комбинации горизонтальных ребер, которые позволяют перейти из одного состояния в другое, разбиваются на две части: когда обратное горизонтальное ребро задействовано при построении цикла, и когда оно не задействовано.

Если обратное горизонтальное ребро не задействовано при построении цикла, переход из некоторого состояния s осуществляется также, как в случае решетки. Если же задействовать обратное ребро, может возникнуть ситуация, когда «левый» (p) и «правый» (q) конец цепи из прошлого, соответствующей дуге (p, q), поменяются местами. То есть после выполнения шага

в будущее с участием обратного ребра, может возникнуть новая цепь, соответствующая дуге (p', q'), в которой $p' > q'$. Но поскольку цепи неориентированные, данная ситуация не влияет на способ кодирования состояний. Можно поменять концы дуг в состоянии местами, и получится дуга (q', p'), не противоречащая принятому способу кодирования, в котором требуется истинность условия $q' < p'$.

Например, пусть состояние $s =$ `(0)(0)`. Пусть при переходе задействовано обратное горизонтальное ребро и еще некоторые ребра так, как это указано на рис. 9(а). Оно соединило концы разных цепей, а другие концы оказались неправильно ориентированными, что можно представить как «состояние» `0)00(0`. Развернув скобки, которые потеряли свои пары, в обратную сторону, получим «правильное» состояние `0(00)0`. Похожая ситуация изображена на рис. 9(б).



(а) (б)

Рис 9. Примеры переходов между состояниями в случае цилиндра

Таким образом, при построении цикла на решетке и цилиндре принят один и тот же способ кодирования состояний — `ПСПРН` длиной m. Это дает возможность с единообразных позиций описать метод динамического программирования для подсчета цепей на решетке и цилиндре. Но прежде, чем приступить к описанию метода, необходимо более подробно рассмотреть вопрос определения переходов между состояниями.

4.1. Переходы между состояниями

Все состояния и всевозможные переходы между ними образуют псевдограф. Для применения метода матрицы переноса можно построить матрицу смежности такого псевдографа и возвести ее в нужную степень. Метод, описанный в настоящей работе, не нуждается в построении матрицы переноса, однако требует способа построения всех состояний, в которые можно перейти из фиксированного состояния s.

Список всех состояний, в которые можно перейти из состояния s, в дальнейшем будем обозначать `To`(s). Пусть $s =$ `((00))`, тогда `To`$(s) =$ `(((0)))`, `00()()`, `(00())`, `(0()0)`, `(()00)`, `00(00)`, `()()00`, `(00)00`. Построение такого списка требует перебора всех 2^{m-1} возможностей установить горизонтальные ребра в процессе шага в будущее. Однако не все эти способы следует перебирать при практической реализации. Существует 3 случая, упомянутых в разделе 2.2, когда выбранная комбинация горизонтальных ребер недопустима. Учет этих случаев не избавляет от перебора различных вариантов установки горизонтальных ребер, но позволяет отсечь заведомо неприемлемые из них. Для примера, на решетке $P_{10} \times P_n$ полный перебор потребует рассмотрения $2^9 = 512$ комбинаций, тогда как допустимых комбинаций не больше 89. Другой пример, для $m = 16$ из $2^{15} = 32\,768$ комбинаций заслуживают внимания лишь 1 597. В общем случае, мы заметили, что для $m \leq 22$ вместо 2^{m-1} комбинаций необходимо перебирать не более f_m, где f_m — число Фибоначчи с номером m ($f_0 = 1$, $f_1 = 1$).

Рассмотрим более подробно на примерах случаи недопустимых комбинаций горизонтальных ребер.

Один из случаев (рис. 10 слева) состоит в том, что нельзя замыкать какую-либо из цепей в цикл. Если допустить такое замыкание, описываемый ниже метод подсчета будет считать количество 2-факторов графа.

Второй случай запрещает соединение в одной вершине более чем двух ребер (рис. 10 по центру), это правило вместе с первым гарантирует, что в конечном итоге получится именно цикл — связный регулярный подграф со степенью вершин, равной 2.

Третий случай возникает в связи с тем, что цикл должен получиться гамильтоновым, поэтому незадействованных в построении цикла вершин быть не должно (рис. 10 справа). Отмена этого правила повлечет за собой подсчет количества всех возможных циклов, а не только гамильтоновых.

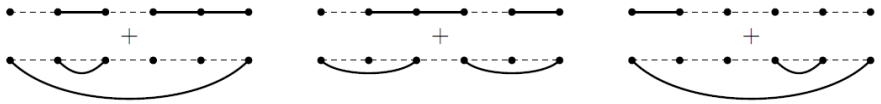


Рис 10. Три случая недопустимых переходов

Из перебора всех возможных комбинаций следует, что три описанных случая полностью исчерпывают запрещенные переходы. Соблюдение трех правил гарантирует гамильтоновость цикла и то, что в процессе построения цикла будут использоваться только допустимые состояния, а недопустимые вообще не рассматриваются.

В случае цилиндра необходимо перебрать 2^m возможностей установить горизонтальные ребра. При этом для цилиндров характерны те же случаи недопустимых комбинаций, что и для решетки. Таким образом, для цилиндров характерно такое же сокращение перебора комбинаций горизонтальных ребер, что и для решетки.

Списки $\mathtt{To}(s)$ для решетки и цилиндра могу быть разными, поэтому разным может оказаться и множество состояний для циклов в этих графах. Однако благодаря единообразной схеме кодирования состояний, описанный в следующем разделе метод работает одинаково не зависимо от вида сеточного графа.

4.2. Метод динамического программирования

В этом разделе предлагается схема метода матрицы переноса, но без явного хранения самой матрицы. Название метода обусловлено тем, что по сути он представляет собой динамический процесс, в котором окончательный ответ к задаче складывается из ответов для некоторых подзадач. Подобная схема, где матрица переноса явно не используется, приводится и в работе [16] (но в словесной форме), с той лишь разницей, что автор указанной работы предлагает иной способ построения и кодирования состояний, более затратный по использованию памяти.

В [16] предлагается выполнять разрез по ломанной линии, а не по прямой, что влечет дополнительные затраты памяти (приблизительно в m раз по сравнению с нашим методом), связанные с хранением не только положения линии разреза, но и положения ее излома. Так, например, автор указанной работы ограничился решетками шириной $m = 20$ из-за того, что следующая по ширине решетка требовала рассмотрения больше 10^8 состояний. Наша схема кодирования для решеток шириной $m = 21, 22$ требует меньшего количества состояний, а для цилиндров с тем же периметром около $6{,}5 \cdot 10^6$ состояний (см. табл. 1, с. 490).

Для наглядности мы предлагаем схему метода динамического программирования на псевдокоде (рис. 11) и комментируем ее ниже.

В схеме используются множества Q и H. В Q хранятся состояния и число способов получить их на шаге $k-1$. В H хранятся состояния и число способов получить их на шаге k. Элементы этих множеств будут иметь вид (s,d), где s — состояние, а d — число способов построить его. Названия Q и H связаны с тем, что первое множество удобно реализовывать в программе в виде очереди, а второе — в виде хэш-таблицы (для быстрой проверки существования в H нужного элемента).

```
Improved–Transfer–Matrix(m, n)
 1  Q ← {(0^m, 1)}
 2  for k = 1 to n
 3        do H ← ∅
 4           for ∀ (s, d) ∈ Q
 5                 do T ← To(s)
 6                    for ∀ t ∈ T
 7                          do if ∃ e : (t, e) ∈ H
 8                                then H ← (H \ {(t, e)}) ∪ {(t, e + d)}
 9                                else H ← H ∪ {(t, d)}
10           Q ← H
11           if ∃ x : (0^m, x) ∈ Q
12             then Answer[k] ← x
13                  Q ← Q \ {(0^m, x)}
14             else Answer[k] ← 0
```

Рис 11. Схема метода динамического программирования на псевдокоде

Строка 1 выполняет инициализацию. На нулевом слое, когда построение цикла еще не началось, строится начальное состояние одним способом. Оно помещается в очередь Q.

Цикл в строках 2–14 перебирает один слой графа за другим с помощью переменной k. Тело этого цикла начинается с инициализации множества H, которое сначала должно быть пустым. В строке 4 для каждого элемента (s,d) множества Q выполняются операции строк 5–9. В строке 5 помощью функции $\mathrm{To}(s)$ символически обозначено получение из s всех состояний, в которые из него можно перейти (как в разделе). Элемент T в строках 5–6 является списком, а не множеством, то есть в нем могут быть одинаковые элементы.

Для каждого полученного из s состояния t (строка 6) мы проверяем, встречалось ли такое состояние ранее, то есть находится ли оно во множестве H и сколькими способами (e) оно было получено (строка 7). Если такое состояние встречалось ранее, то к числу способов e его построить, добавляется число способов d, построить состояние s (строка 8). Иначе (строка 9), состояние t и число способов d первый раз (для текущего k) добавляются в H.

После того, как все элементы множества Q обработаны, в него записываются все элементы полученного множества H (строка 10), чтобы затем начать обработку следующего слоя графа. Но перед тем, как снова перейти к строке 3, необходимо проверить, было ли на текущем шаге k достигнуто финальное состояние (строка 11).

Если финальное состояние было достигнуто, то число способов x достигнуть его и является числом циклов на решетке $P_m \times P_k$ (или цилиндре $C_m \times P_k$). В этом случае финальное состояние нужно удалить из множества Q (строка 13), иначе на следующем шаге оно будет воспринято как начальное, поскольку также кодируется в программе последовательностью нулей.

Если финальное состояние не было достигнуто на шаге k, значит число циклов в этом случае равно 0 (строка 14). После завершения процедуры, число $\mathrm{Answer}[i]$ будет равно числу циклов на решетке (цилиндре) размером $m \times i$ $(i = 1, \ldots, n)$.

Таким образом, достоинством этого подхода является экономия оперативной памяти, однако недостаток заключается в том, что для каждого состояния s необходимо каждый раз снова определять, какие состояния t из него можно построить.

Параллельные вычисления позволяют сократить время расчетов, которое является платой за экономию памяти. Один из возможных методов распараллеливания предложенного метода приводится в работе [2].

4.3. Оптимизация

Кроме метода динамического программирования в [2] указывается один из способов оптимизации алгоритма, связанный с удалением недостижимых и отождествлением идентичных состояний. Так, состояние `0(0)` никогда не возникнет при построении цикла на решетке $P_4 \times P_n$. Чем шире решетка, тем больше недопустимых состояний можно отыскать и удалить из рассмотрения. Впервые внимание теоретическому способу обнаружения недостижимых состояний для решеток было уделено в работе [7]. Если число возможных состояний для $m = 4$ равно 9, то с учетом предложенного метода автор указанной работы получает 6 достижимых состояний (он не учитывает нулевое состояние). Для $m = 10$ оптимизация дает бóльший эффект: возможных состояний 2 188, а достижимых — 1 117.

В работе [2] впервые предложен метод учета идентичных состояний. Состояния, которые получаются друг из друга симметричным отражением можно отождествить. Например, состояния `(())()` и `()(())` на решетке $P_6 \times P_n$ можно считать одинаковыми и хранить только одно из них. Для $m = 10$ данный способ дает 607 допустимых состояний, что почти вдвое меньше, чем в работе [7].

В случае с цилиндром идентичными считаются состояния, которые получаются друг из друга циклическим сдвигом. Например, на цилиндре $C_5 \times P_n$ это состояния `0()()`, `()()0`, `(()0)`, `()0()`, `(0())`. Так, для $m = 10$ число достижимых состояний на цилиндре будет равно 123, что значительно меньше, чем 607 состояний на решетке. С учетом такой оптимизации, общее количество состояний для решеток и цилиндров для $m \le 22$ приводится в табл. 1.

Таблица 1. Сравнение количества возможных состояний с числом достижимых состояний на решетке и цилиндре

m	$\mathcal{M}_m$	К-во состояний	
		$P_m \times P_n$	$C_m \times P_n$
3	4	3	2
4	9	6	3
5	21	12	5
6	51	23	9
7	127	62	19
8	323	109	26
9	835	365	95
10	2 188	607	123
11	5 798	2 355	528
12	15 511	3 774	619

m	$\mathcal{M}_m$	К-во состояний	
		$P_m \times P_n$	$C_m \times P_n$
13	41 835	16 020	3 219
14	113 634	25 188	3 600
15	310 572	113 198	20 714
16	853 467	176 061	21 917
17	2 356 779	821 923	138 635
18	6 536 382	1 270 562	141 186
19	18 199 284	6 097 041	957 858
20	50 852 019	9 387 784	938 155
21	142 547 559	46 013 564	6 788 019
22	400 763 223	70 652 188	6 422 928

Можно подходить к применению метода матрицы переноса в рамках теории автоматов, где известны стандартные схемы их минимизации, однако авторам [6] не удалось добиться с их помощью более существенного изменения размера матрицы переноса. Например, для $m = 12$ число состояний удалось уменьшить до 4 828, что хотя и намного лучше, чем в работе [7] (7 280), но хуже нашего метода, учитывающего симметрию (3 774), а для случая цилиндров, в которых учитываются циклические сдвиги, число состояний снижается еще более радикально (619). Применение более сложных методов оптимизации конечных автоматов, таких как

отождествление идентичных состояний по принципу одинакового числа способов их достигнуть, резко усложняет вычисления, особенно для больших значений m, когда автомат не умещается целиком в памяти компьютера.

Проблема быстрого роста количества циклов не исчезает после всех сделанных оптимизаций, поэтому может оказаться удобным вместо хранения длинных чисел использовать модулярную арифметику. Программа вычисляет количество циклов по модулю нескольких подряд идущих простых чисел: $2^{31}-1$, $2^{31}-19$, $2^{31}-61$, ... Затем ответ восстанавливается с помощью Китайской теоремы об остатках. Для этого можно использовать любую систему компьютерной алгебры, поддерживающую длинные числа. Первым числом в последовательности простых чисел выбрано $2^{31}-1$, так как оно является максимальным из тех, что умещаются в 32 бита со знаком, но при этом сумма двух таких чисел не покидает знакового 32-битового диапазона. Модулярная арифметика также использовалась в [16].

Таким образом, сворачивание решетки в цилиндр не меняет способа кодирования состояния и не приводит к усложнению подсчетов, а, напротив, позволяет особо выделить новое свойство графа, упрощающее подсчет количества циклов в нем. Это свойство, связанное с циклическими граничными условиями, накладываемыми на левую и правую границы, помогает значительно сократить число рассматриваемых состояний. Однако наложение циклических граничных условий на верхнюю и нижнюю границы, превращающих цилиндр $C_m \times P_n$ в тор $C_m \times C_n$, напротив, значительно усложняет схему вычислений.

5. КОДИРОВАНИЕ СОСТОЯНИЙ НА ТОРАХ $C_m \times C_n$

В случае решетки можно строить цикл, начиная от нижней границы. Двигая линию разреза снизу вверх, мы достраивали оба конца каждой цепи на один шаг в будущее и следили за тем, чтобы на последнем шаге все получившиеся цепи соединились в одну, образуя цикл. При этом способы начала построения цикла не влияют на варианты его завершения, то есть финальное состояние, которое кодировалось строкой из нулей, никак не зависело от начального состояния, которое кодировалось точно также. Аналогичное можно сказать и о цилиндре, если развернуть его на плоскости и выбрать в качестве исходной границы нижнюю.

У тора нет границы, выбрав которую в качестве исходной, можно было бы начать построение цикла также, как в случае решетки и цилиндра. Такую границу можно создать искусственно, развернув тор на плоскости как на рис. 2(в), с. 477.

В отличие от решетки и цилиндра, в которых построение цикла начинается с нулевого состояния, в случае тора построение должно начинаться с выбора вертикальных обратных ребер, которые будут участвовать в построении цикла. При этом финальное состояние будет зависеть от этого начального выбора и от того, как цепи, проходящие через выбранные вертикальные обратные ребра, продолжались в будущее. Иными словами, если в случае решетки и цилиндра форма цикла у начального слоя не может влиять на способ завершения построения всего цикла, то в случае торов — может. Пример такой ситуации проиллюстрирован на рис. 12.

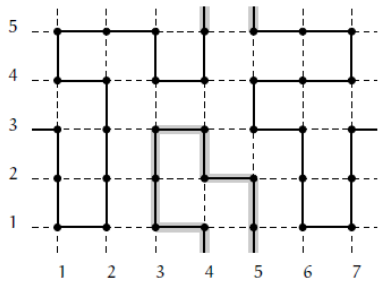


Рис 12. Как нижняя часть цикла может влиять на верхнюю

На этом рисунке изображен тор $C_7 \times C_5$ и цикл в нем. В качестве обратных вертикальных ребер, которые будут участвовать в построении цикла, были выбраны ребра 4 и 5. Цепи, которые проходят по этим ребрам, продолжаясь в будущее, соединились на третьем уровне (этот участок цикла выделен на рисунке серым цветом). Эта информация играет важную роль при выяснении того, могут ли полученные на последнем слое цепи объединиться в цикл.

По этой причине состояние в случае тора кодируется иначе, чем в случае решетки и цилиндра. Необходимо хранить не только информацию о цепях, проходящих через текущий уровень графа, но и информацию о цепях, начинающихся на самом нижнем уровне.

Поясним на примере. На рис. 13 изображен тор $C_8 \times C_{10}$ и цикл на нем. Справа изображена последовательность закодированных состояний, которые образуются при построении этого цикла снизу вверх; смысл этих кодов будет поясняться ниже. Предположим, что цикл построен лишь до 5-го слоя графа (до пунктирной линии разреза). Данный фрагмент тора ($C_8 \times C_5$) отражен на рис. 14 сверху. Рассмотрим, как символически кодируется информация о состоянии недостроенного цикла на этом фрагменте.

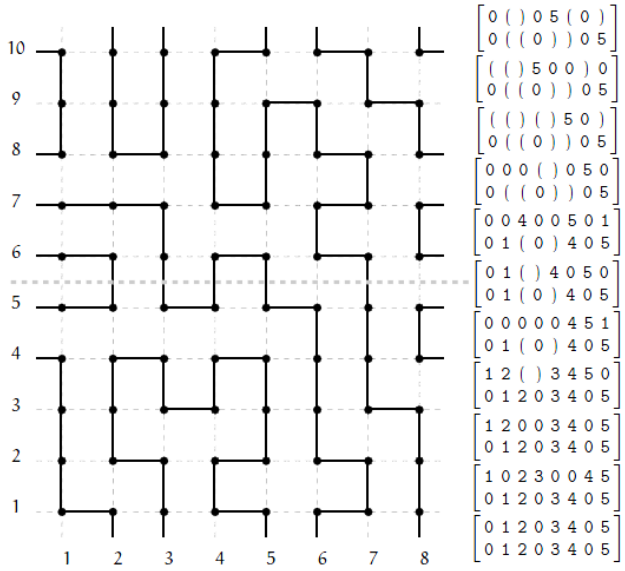


Рис 13. Цикл на торе $C_8 \times C_{10}$ и состояния, образующие его

На рис. 14 снизу изображена часть тора, если из нее удалить слои со второго по четвертый, оставив лишь первый и пятый. Такой тор будем называть *сжатым*. Символически, дугообразные линии на изображении сжатого тора, концы которых находятся на одном уровне, соответствуют цепям, проходящим по не сжатому тору. Например, на рис. 14 снизу дугообразная линия соединяет вершины 3 и 5 первого слоя. Это означает, что на рассматриваемом участке тора $C_8 \times C_5$ есть цепь, начинающаяся в вершине 3 первого слоя и заканчивающаяся в вершине 5 этого же слоя. Информацию о том, как именно данная цепь проходит по вершинам тора хранить не требуется. Линии, соединяющие на сжатом торе нижний слой с верхним соответствуют цепям которые берут свое начало в первом слое и каким-то образом достигают последнего слоя рассматриваемого фрагмента. Ниже будет показано как этот способ кодирования позволяет определить, построен ли цикл.

Таким образом, любое состояние, возникающее при построении цикла на торе, характеризуется тем, как цепи, постепенно достраиваемые снизу вверх, проходят между первым уровнем и текущим. Каждое состояние может содержать цепи двух типов. Цепи *первого типа* обоими концами касаются одного уровня и проходят через прошлое, они на изображении состояния символически обозначаются дугообразной линией. Цепи *второго типа* начинаются у первого

слоя и заканчиваются у текущего. В символическом изображении состояния эти цепи обозначаются линиями, которые соединяют вершины нижнего слоя и верхнего на сжатом торе.

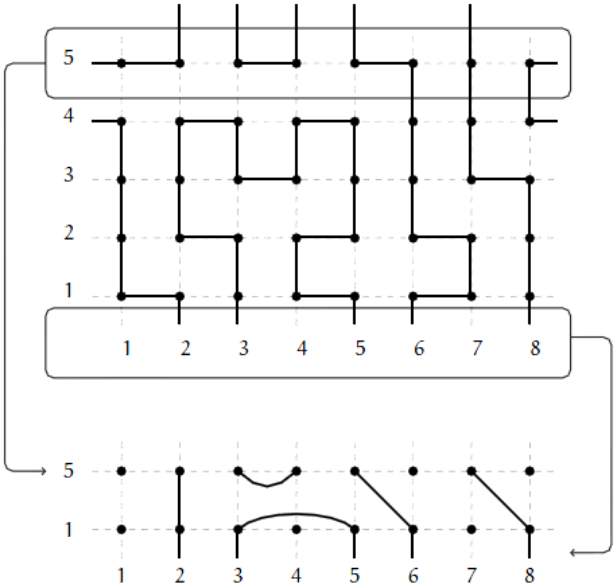
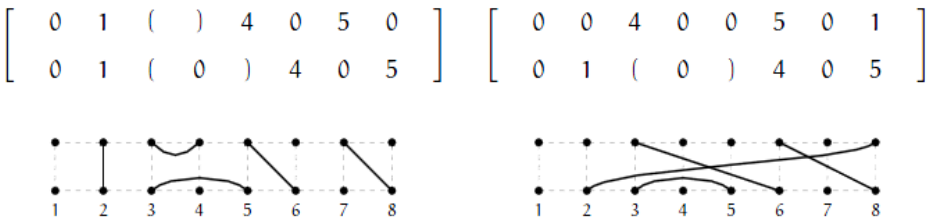


Рис 14. Сжатие цикла на торе

Указанный способ симоволического изображения состояний, возникающих при построении цикла на торе, удобен при рассмотрении способов перехода между состояниями. В иных случаях может оказаться удобным записывать состояния в виде специального кода, состоящего из двух строк. Преобразование символического изображение в такой код рассмотрим на примере, изображенном на рис. 15(а).



(а) (б)

Рис 15. Примеры кодирования состояний из рис. 13: (а) для слоя $k = 5$; (б) для слоя $k = 6$

Скобки соответствуют цепям первого типа, а цифры — концам цепей второго типа. Одна и та же цифра на нижнем слое и на верхнем соответствует двум концам одной и той же цепи. Стоит обратить внимание (рис. 13) на то, что нижняя строка состояния в самом начале построения цикла может содержать только нули и цифры; постепенно, переходя к следующим состояниям, какие-то пары цифр могут превращаться в скобки, но нули в нижней строке всегда будут оставаться на тех местах, где они были сначала. После изменения цифр на скобки, нумерацию менять не обязательно.

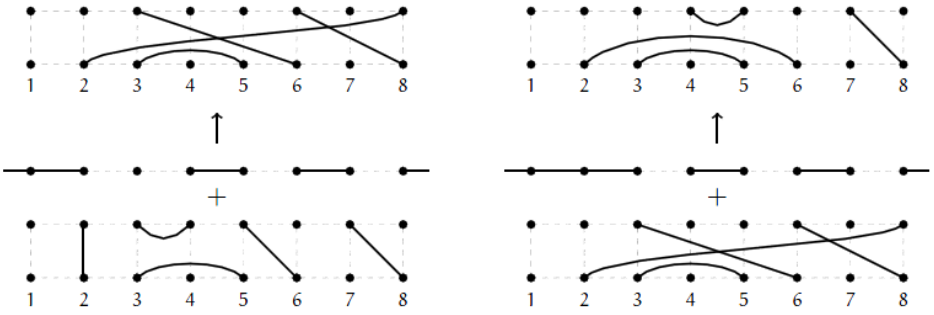
Еще один пример проиллюстрирован на рис. 15(б). Данное состояние соответствует слою $k = 6$ на рис. 13. Здесь также показано, что на символическом изображении состояния не обязательно указывать циклические граничные условия в виде коротких отрезков. Их отсутствие не мешает видеть общую структуру строящегося цикла, но позволяет выполнить рисунок более компактно.

5.1. Переходы между состояниями

Для того, чтобы проиллюстрировать способ перехода от одного состояния во все остальные, вернемся с нова к символическому способу кодирования состояний. Рассмотрим способы перехода на примере двух состояний, изображенных ранее на рис. 15.

Для перехода в другие состояния потребуется перебрать 2^m способов установить горизонтальные ребра. Но помимо тех случаев, которые были описаны для решетки и цилиндра (случаи, связанные с цепями первого типа), для торов есть и другие, которые касаются цепей второго типа.

Если цепь второго типа соединяется с цепью первого типа, то она становится продолжением цепи второго типа. Один из ее концов должен получить соответствующую цифру. Пример такого перехода изображен на рис. 16(а). Формально говоря, цепь первого типа исчезает.

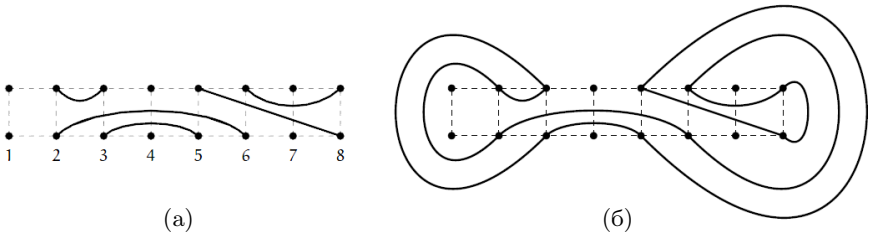


(а) (б)

Рис 16. Примеры переходов из рис. 13:
(а) от слоя $k = 5$ к слою $k = 6$; (б) от слоя $k = 6$ к слою $k = 7$

Если цепь второго типа соединяется с цепью второго типа, то обе они исчезают (не переходят в будущее). Обе цепи второго типа, объединившись, превращаются в одну цепь первого типа. Такая ситуация изображена на рис. 16(б).

Еще одно отличие в построении цикла на торе от построения цикла на цилиндре состоит в том, что, иначе определяется финальное состояние. В случае цилиндра финальным было состояние, все цепи первого типа у которого соединялись правильным образом, образуя нулевое состояние. Ненулевое финальное состояние определяется следующим образом. Рассмотрим состояние на рис. 17(а), оно соответствует тому, что получается на рис. 13, когда цикл уже построен.



(а) (б)

Рис 17. Финальное состояние из рис. 13:
(а) Изображение состояния; (б) Проверка того, что состояние является финальным

В этом состоянии нужно соединить обратными ребрами все входящие в него дуги, то есть цепи первого и второго типа, как показано на рис. 17(б). Если после этого получается цикл, то состояние является финальным.

5.2. Метод динамического программирования

Схема метода динамического программирования для случая торов мало чем отличается от рассмотренной в разделе 4.2 схемы для случая решеток и цилиндров. В программу, псевдокод которой приводится на рис. 11, следует внести некоторые изменения.

Вместо операции в строке 1 программы, которая добавляет в очередь начальное состояние, требуется операция, добавляющая в очередь все возможные 2^m начальных состояний. Число способов построить начальное состояние по определению полагается равным 1.

Любое начальное состояние на торе получаются путем выбора некоторого набора обратных вертикальных ребер. Например на рис. 13 начальное состояние имеет вид $\left[\begin{smallmatrix}0&1&2&0&3&4&0&5\\0&1&2&0&3&4&0&5\end{smallmatrix}\right]$. В нем выбраны вертикальные обратные ребра с номерами 2, 3, 5, 6 и 8. Это лишь одно из 2^8 начальных состояний.

Строки 11–13 в программе на псевдокоде должны быть изменены. Вместо проверки существования нулевого состояния, необходимо отыскать количество финальных состояний (s, x_s) в очереди Q. Число способов оказаться в них $\sum_s x_s$ должно быть добавлено к переменной $Answer[k]$. Нулевые состояния (кодируемые строками из нулей) должны быть затем удалены, так как они соответствуют циклам на цилиндрах и могут быть восприняты при следующей итерации как начальные.

5.3. Количество состояний

Оценим верхнюю границу на число всех возможных состояний, которые могут возникнуть при подсчете циклов на торе $C_m \times C_n$. Здесь же будет показано, что не все 2^m начальных состояний можно рассматривать при вычислениях.

Всего имеется 2^m способов выбрать множество обратных ребер, соединяющих верхнюю и нижнюю часть тора. Забудем на время о том, что обратные ребра в коде состояния записываются разными цифрами и будем использовать битовую маску длиной m из 0 и 1. Ноль означает, что обратное ребро не задействовано, а единица — задействовано. В силу циклических граничных условий, некоторые из этих способов выбрать множество обратных ребер можно считать одинаковыми, так как они дают одно и то же число циклов. Например, при $m = 4$, способы `1100`, `1001`, `0011` и `0110` могут быть получены один из другого путем циклического сдвига, то есть можно посчитать количество циклов, которые задействуют лишь первое и второе ребра (`1100`), а затем умножить результат на 4.

Ответом на вопрос о том, сколько всего существует последовательностей из нулей и единиц, которые не могут быть получены друг из друга циклическим сдвигом дает задача об ожерельях — классическая задача перечислительной комбинаторики.

Ожерелье может быть составлено из n бусин черного и белого цвета. Причем черных из них ровно k. Чему равно число различных ожерелий $\mathcal{N}_n^k$?

На этот вопрос отвечает теорема, называемая теоремой Редфилда — Пойа, одно из следствий которой может быть сформулировано в виде формулы:

$$\mathcal{N}_n^k = \frac{1}{n} \sum_{d|(n,k)} \phi(d) \binom{n/d}{k/d},$$

где запись (n, k) означает наибольший общий делитель n и k, а ϕ — функция Эйлера. Число ожерелий значительно меньше числа сочетаний, которые было бы необходимо перебирать в том случае, если бы мы не учли циклических сдвигов. Так, $\mathcal{N}_{10}^5 = 26$, а $\binom{10}{5} = 252$.

Указанная теорема сразу отвечает на вопрос о том, сколькими различными способами могут быть выбраны обратные ребра при построении цикла на торе. Зафиксируем некоторый такой

выбор из i обратных ребер. Любые состояния, порождаемые данным выбором будут в нижней строке состояния содержать $m-i$ нулей на строго определенных местах, которые никогда не будут меняться. Цифры от 1 до i всегда будут находиться в нижней строке в порядке возрастания, но некоторые пары цифр могут быть заменены на соответствующие друг другу скобки.

Пусть j — число пар цифр из нижней строки, которые объединились в пару соответствующих друг другу скобок. Это может произойти $\mathcal{C}_j\binom{i}{2j}$ $(j=0,1,\ldots,\lfloor i/2\rfloor)$ способами, где $\mathcal{C}_j$ — число Каталана.

Если j пар цифр объединились в нижней строке, то это означает, что в верхней строке их осталось ровно $i-2j$. Таким образом, в верхней строке осталось $m-(i-2j)$ свободных мест для расположения в них концов цепей первого типа (правильных скобочных последовательностей, разряженных нулями). Это можно сделать $\mathcal{M}_{m-(i-2j)}\binom{m}{i-2j}$ способами.

Последнее, что нужно учесть — цифры из нижней строки могут оказаться в верхней строке в другом порядке, не в порядке возрастания. Но, принимая во внимание структуру графа, эти цифры будут сохранять свой порядок с точностью до циклического сдвига, а таких сдвигов может быть не больше количества самих цифр, перешедших в верхнюю строку, то есть $i-2j$.

Если же ни одна цифра не перешла в верхнюю строку, то число состояний определяется числом Моцкина $\mathcal{M}_m$, а такое может случиться лишь при четном i и при $j=i/2$ (в этом случае $i-2j=0$). Этот случай рассматривается отдельно.

Суммируя сказанное, имеем следующую формулу

$$\mathcal{S}_m=\sum_{i=0}^{m}\mathcal{N}_m^i\sum_{j=0}^{\lfloor i/2\rfloor}(i-2j)\cdot\mathcal{C}_j\cdot\mathcal{M}_{m-(i-2j)}\cdot\binom{i}{2j}\cdot\binom{m}{i-2j}+\mathcal{M}_m\sum_{i=0}^{\lfloor m/2\rfloor}\mathcal{N}_m^{2i}.$$

Детальный анализ поведения последовательности $\left(\mathcal{S}_m\right)$ представляется весьма трудоемким с точки зрения цели этого анализа — выяснения скорости роста чисел $\mathcal{S}_m$ при $m\to\infty$. Эту задачу можно решить статистическими методами, например, методом отношений. Расчет величин $\mathcal{S}_m/\mathcal{S}_{m-1}$ для $m=4,5,\ldots,1000$, показал их устойчивое стремление к числу 16, что позволило оценить скорость роста порядком $O(16^m)$.

5.4. Оптимизация

Табл. 2 показывает, насколько теоретическая оценка на величину $\mathcal{S}_m$ расходится к количеством допустимых состояний.

Таблица 2. Верхняя оценка и число допустимых состояний для тора $C_m\times C_n$

m	$\mathcal{S}_m$	Число состояний	
		Всех достижимых	На четн. / неч. слоях
3	41	25	22
4	308	128	88
5	2 234	1 010	820
6	22 632	5 548	3 565
7	201 267	55 828	44 005
8	2 150 828	294 026	181 746
9	22 346 526	3 459 934	2 689 084
10	249 907 248	17 290 669	10 456 654
11	2 765 454 456	$>100\,000\,000$	$>100\,000\,000$

Причина расхождения в том, что оценка была получена из предположения, что, во-первых, все входящие в рассуждения выше конфигурации являются возможными, но не обязательно допустимыми, а во-вторых, что все конфигурации могут быть построены на каждом слое. На самом деле, есть три причины, по которым теоретическая оценка расходится с реальным числом состояний, которые необходимо одновременно хранить в памяти в процессе вычислений.

Рассмотрим эти причины на примере всех состояний для минимально возможного значения $m = 3$. Формула для $\mathcal{S}_m$ говорит, что $\mathcal{S}_3 = 41$. Изобразим все возможные состояния на рис. 18.

Состояния на этом рисунке пронумерованы от 0 до 40. Некоторые номера помечены символом $\times$, 0, 1 или $\circlearrowleft$. Первая причина, по которой реальное число состояний меньше, чем теоретическое, заключается в том, что не все состояния достижимы. Недостижимые состояния отмечены символом $\times$, их в данном примере всего два. Вторая причина — циклические сдвиги, которые применяются для оптимизации расчетов. Некоторые состояния получаются одно из другого путем циклического сдвига — они отмечены символом $\circlearrowleft$. Так, состояния 23, 30 и 37 получаются одно из другого, циклическим сдвигом, поэтому два из них (пусть, 30 и 37) можно считать лишними. Аналогичное можно сказать про тройку состояний (24, 31, 35) или (1,2,3). Если убрать все недостижимые состояния и оставить по одному состоянию из группы тех, что получаются друг из друга циклическим сдвигом, то получится число состояний, указанное в таблице ниже в третьей колонке.

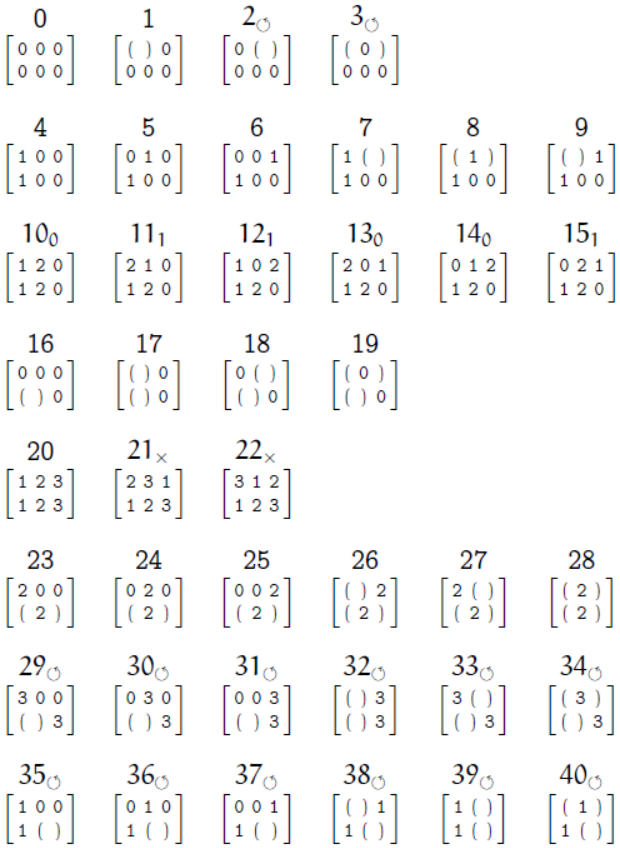


Рис 18. Все возможные состояния, возникающие на торе $C_3 \times C_n$

Но число состояний, которое нужно одновременно хранить в памяти еще меньше по третьей причине: некоторые состояния достижимы только на четных слоях графа (помечены символом 0), а другие — на нечетных (помечены символом 1). На практике наблюдается, что таких

состояний поровну для всех $m = 3, \ldots, 10$, поэтому на четных и нечетных шагах построения цикла число состояний одно и то же (указано в четвертой колонке), но оно меньше, чем число всех используемых состояний (что указано в колонке 3).

Полученная оценка $\mathcal{S}_m$ вполне неплохо объясняет, почему подсчет циклов на торах значительно труднее, чем подсчет на решетках и цилиндрах. Расхождение данной оценки с практическим результатом примерно такое же, как в случае цилиндров.

6. ОБСУЖДЕНИЕ РЕЗУЛЬТАТОВ

В работе представлены разнообразные оптимизации, связанные со схемой кодирования состояний, которые позволили сделать несколько шагов вперед по сравнению с предыдущими исследователями. Отметим кратко причину нашей остановки на $m = 22$ для решеток цилиндров и $m = 10$ для торов. Имеющиеся у нас средства вычислений позволяли одновременно хранить в памяти компьютера до 4 ГиБ данных, хотя вычислительных ядер с частотой 2,67 ГГц в количестве 64 было вполне достаточно. На такой вычислительной машине рассчеты для решетки $P_{22} \times P_{100}$ можно было проводить сразу по трем простым числам в течении 30 часов, а для правильного восстановления ответа требовалось не более 39 модулей. То есть необходимо было запускать программу 13 раз, ожидая, в общей сложности, 390 часов. Но причина остановки на значении $m = 22$ заключалась в том, что для $m = 23$ памяти оказалось недостаточно даже для проведения вычислений по одному модулю, так как число состояний превысили 10^8 (точное его количество осталось неизвестным), аналогичная ситуация возникла в случае торов для $m = 11$. В случае цилиндров для $m > 22$ сложность состояла в отсутствии эффективного способа вычислений переходов между состояниями, который требует времени порядка $O(f_m)$, что не было критичным для $m = 22$. Таким образом, для дальнейших расчетов предложенным методом потребуется не только иметь больше 4 ГиБ оперативной памяти, но и разработать эффективный метод перехода между состояниями, либо увеличить количество вычислительных ядер.

Несмотря на эти ограничения в имеющейся архитектуре, удалось посчитать достаточно длинные последовательности для количества циклов, чтобы вывести ряд новых рекуррентных соотношений с постоянными коэффициентами для решеток шириной до $m = 14$, цилиндров с периметром до $m = 16$ и торов с меньшим периметром до $m = 8$. Порядки этих рекуррентных соотношений записаны в следующей таблице (символом * отмечены результаты, полученные впервые автором работы)

Таблица 3. Порядки рекуррентных соотношений

m	3	4	5	6	7	8	9	10
$P_m \times P_n$	2	4	6	14	36	66	208	*346
$C_m \times P_n$	1	2	3	*7	*12	*20	*51	*74
$C_m \times C_n$	*6	*28	*84	*257	*856	*2 785		

m	11	12	13	14	15	16
$P_m \times P_n$	*1 342	*2 086	*8 958	*13 523		
$C_m \times P_n$	*246	*303	*1 320	*1 514	*7 936	*8 363

Подробное рассмотрение методов вывода этих соотношений и их анализ заслуживает отдельной статьи.

Все рекуррентные соотношения доступны на сайте автора [18].

СПИСОК ЛИТЕРАТУРЫ

1. Valiant L.G. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 1979, vol. 8, no. 3. pp. 410–421.
2. Караваев А.М. Усовершенствованный метод матрицы переноса для подсчета гамильтоновых циклов на прямоугольных решетках и цилиндрах. *Актуальные проблемы гуманитарных и естественных наук*, 2011, №4(27), стр. 16–24.
3. Караваев А.М., Перепечко С.Н. Подсчет гамильтоновых циклов на семействах торов $C_m \times C_n$. *XVII международная конференция по вычислительной механике и современным прикладным программным системам (ВМСППС'2011), Алушта.* М : Изд-во МАИ-ПРИНТ, 2011, стр. 243–245.
4. Tošić R., Bodroza O., Kwong Y.H.H., Straight H.J. On the number of hamiltonian cycles of $P_4 \times P_n$. *Indian Journal of pure and applied Mathematics*, 1990, vol. 21, no. 5, pp. 403–409.
5. Klein D.J. Asymptotic distributions for self-avoiding walks constrained to strips, cylinders, and tubes. *Journal of Statistical Physics*, 1980, vol. 23, no. 5, pp. 561–586.
6. Stoyan R., Strehl V. Enumeration of Hamiltonian Circuits in Rectangular Grids. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 1996, vol. 21, pp. 109–127.
7. Kloczkowski A., Jernigan R.L. Transfer matrix method for enumeration and generation of compact self-avoiding walks. I. square lattices. *Journal of Chemical Physics*, 1998, vol. 109, pp. 5134–5146.
8. Schmalz T.G., Hite G.E., Klein D.J. Compact self-avoiding circuits on two-dimensional lattices. *Journal of Physics A: Math. and Gen.*, 1984, vol. 17, pp. 445–453
9. Derrida B. Phenomenological renormalisation of the self-avoiding walk in two dimensions. *Journal of Physics A: Math. and Gen.*, 1981, vol. 14, no. 1, pp. L5–L9.
10. Cloizeaux J., Jannink G. *Polymers in Solution: Their Modelling and Structure.* Oxford : Clarendon Press, 1990.
11. Faase F.J. On the number of specific spanning subgraphs of the graphs $G \times P_n$. *Arc Combinatoria*, 1998, vol. 49, pp. 129–154.
12. Kreweras G. Dénombrement des cycles hamiltoniens dans un rectangle quadrillé. *European Journal of Combinatorics*, 1992, vol. 13, no. 6, pp. 473–476.
13. Kwong Y.H.H., Rogers D.G. A Matrix Method for Counting Hamiltonian Cycles on Grid Graphs. *Eur. J. Comb.*, 1994, vol. 15, pp. 277–283.
14. Jacobsen J.L., Kondev J. Field theory of compact polymers on the square lattice. *Nuclear Physics*, 1998, vol. B532[FS], pp. 635–688.
15. Lundow P.H. Compression of transfer matrices. *Discrete Mathematics*, 2001, vol. 231, pp. 321–329.
16. Jacobsen J.L. Exact enumeration of Hamiltonian circuits, walks and chains in two and three dimensions. *Journal of Physics A: Math. and Theor.*, 2007, vol. 40, pp. 14667–14678.
17. Higuchi S. Field theoretic approach to the counting Hamiltonian cycles of graphs. *Physical Review E*, 1998, vol. 58, no. 1. pp. 128–132.
18. FlowProblem [электронный ресурс]. Copyright © 2008–2011. Artem M. Karavaev. URL: `http://www.flowproblem.ru`.