МАТЕМАТИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ МАШИН, КОМПЛЕКСОВ И КОМПЬЮТЕРНЫХ СЕТЕЙ

DOI 10.54398/20741707_2022_3_44
УДК 519.178/.245:004.94

СРАВНЕНИЕ АЛГОРИТМОВ ГРАССБЕРГЕРА И АХУНЖАНОВА ДЛЯ НАХОЖДЕНИЯ ОСТОВА ПЕРКОЛЯЦИОННОГО КЛАСТЕРА В ЗАДАЧАХ ПЕРКОЛЯЦИИ УЗЛОВ НА КВАДРАТНОЙ РЕШЕТКЕ

Статья поступила в редакцию 19.08.2022, в окончательном варианте – 30.08.2022.

Гордеев Иван Иванович, Астраханский государственный университет им. В. Н. Татищева, 414056, Российская Федерация, г. Астрахань, ул. Татищева, 20а,
кандидат физико-математических наук, ORCID: 0000-0001-5036-4791, e-mail: g2i@mail.ru
Саенко Наталья Сергеевна, Астраханский государственный университет им. В. Н. Татищева, 414056, Российская Федерация, г. Астрахань, ул. Татищева, 20а,
преподаватель, ORCID: 0000-0002-3595-6892, e-mail: saenko.natasha@mail.ru

В данной статье сравнивается алгоритм Грассбергера и алгоритм Ахунжанова для нахождения проводящего остова перколяционного кластера. Поскольку алгоритм Грассбергера был предложен для задач перколяции узлов на квадратной решетке, то сравнение алгоритмов производится именно в этих задачах. Для алгоритма Грассбергера подробно обсуждаются случаи, когда алгоритм некорректно присоединяет к остову висячие части. Также обсуждаются модификации алгоритма Грассбергера, позволяющие сократить количество присоединяемых к остову висячих частей. Затем обсуждается алгоритм Ахунжанова, позволяющий отделить все висячие части от остова. Дается оценка доли узлов, некорректно присоединяемых к остову разными версиями алгоритма Грассбергера. Также дается теоретическая оценка сложности по времени алгоритма Ахунжанова и экспериментальное сравнение времени работы различных версий алгоритма Грассбергера и алгоритма Ахунжанова.

Ключевые слова: идентификация остова, перколяция узлов, двумерная решетка, открытые граничные условия, алгоритмы на графах, алгоритм Грассбергера, алгоритм Ахунжанова

COMPARISON OF GRASSBERGER AND AKHUNZHANOV ALGORITHMS FOR FINDING THE BACKBONE OF PERCOLATION CLUSTER IN PROBLEMS OF SITE PERCOLATION ON SQUARE LATTICE

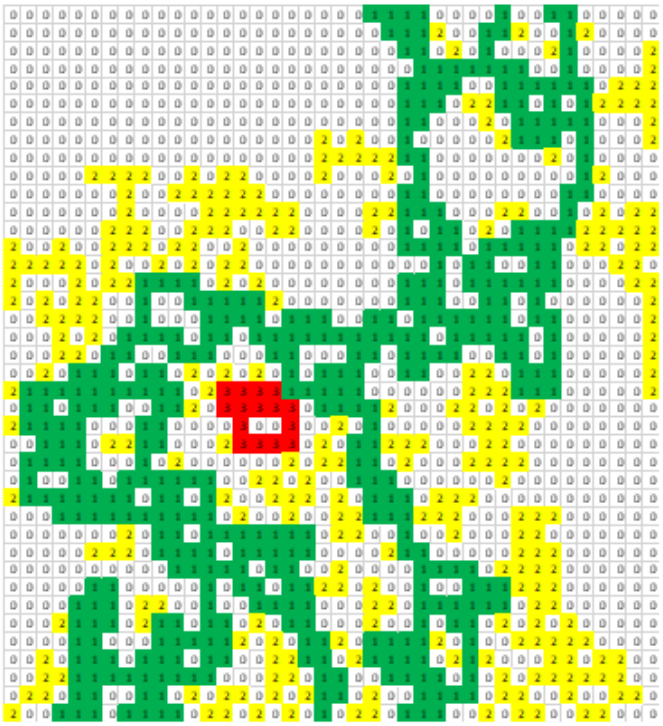
The article was received by the editorial board on 19.08.2022, in the final version – 30.08.2022.

Gordeev Ivan I., Astrakhan State University named after V. N. Tatishchev, 20a Tatishchev St., Astrakhan, 414056, Russian Federation,
Cand. Sci. (Physics and Mathematics), ORCID: 0000-0001-5036-4791, e-mail: g2i@mail.ru
Saenko Natalya S., Astrakhan State University named after V. N. Tatishchev, 20a Tatishchev St., Astrakhan, 414056, Russian Federation,
teacher, ORCID: 0000-0002-3595-6892, e-mail: saenko.natasha@mail.ru

This article compares the Grassberger algorithm and the Akhunzhanov algorithm for finding the conducting backbone of percolation cluster. Since the Grassberger algorithm was proposed for the problems of site percolation on square lattice, the comparison of algorithms is made in these problems. For the Grassberger algorithm, we discuss in detail the cases when the algorithm incorrectly attaches dangling parts to the backbone. We discuss also modifications of the Grassberger algorithm, which allow to reduce the number of dangling parts attached to the backbone. Then Akhunzhanov`s algorithm, which allows separating all dangling parts from the backbone, is discussed. An estimate is given of the proportion of sites incorrectly attached to the backbone by different versions of the Grassberger algorithm. A theoretical estimate of the time complexity of the Akhunzhanov algorithm and an experimental comparison of the runtime of different versions of the Grassberger algorithm and the Akhunzhanov algorithm are also given.

Keywords: backbone identification, site percolation, two-dimensional lattice, open boundary conditions, graph algorithms, Grassberger algorithm, Akhunzhanov algorithm

Graphical annotation (Графическая аннотация)



Пример определения остова перколяционного кластера на квадратной решетке 40x40 (зеленый). Желтым – висячие части. Красным – висячая часть, ошибочно присоединяемая алгоритмом Грассбергера к остову.

An example of determining the core of a percolation cluster on a 40x40 square lattice (in green). In yellow – dangling parts. In red – dangling part, erroneously attached by the Grassberger algorithm to the backbone.

Введение. Задача нахождения проводящего остова относится к одной из задач теории перколяции и является актуальной для различных физических приложений [1, 2].

В теории перколяции обычно рассматриваются случайные графы, для которых ищется путь между двумя вершинами или двумя группами вершин. Обычно предполагается, что вершины из одной группы расположены «с одной стороны графа» при конкретном изображении графа на плоскости или конкретном вложении графа в трехмерное пространство. Если две вершины из разных групп входят в одну компоненту связности, то эту компоненту связности называют перколяционным кластером.

Случайные графы при моделировании перколяции могут генерироваться различными способами. Одним из наиболее популярных способов генерирования случайных графов является рассмотрение периодических решеток, узлы которых соответствуют вершинам графа, а связи между узлами – ребрам графа. Для генерации случайного графа на периодической решетке могут выбираться случайным образом либо связи, тогда говорят о *перколяции связей (bond percolation)*, либо узлы, тогда говорят о *перколяции узлов (site percolation)*. Выбранные для помещения в случайный граф элементы периодической решетки называют *занятыми (occupied)*. В перколяции связей в случайный граф считаются помещенными все вершины, инцидентные занятым ребрам, а в перколяции узлов в случайный граф считаются помещенными все ребра, инцидентные двум занятым вершинам.

С физической точки зрения, наличие перколяционного кластера означает возможность протекания между двумя группами вершин, например, электрического тока, газа или жидкости, если между вершинами из разных групп приложена разность потенциалов (разность давлений). При этом предполагается, что ко всем вершинам одной группы приложен одинаковый потенциал (одинаковое давление). При моделировании протекания электрического тока обычно предполагается, что одинаковый потенциал в одной группе вершин создается за счет подключения всех вершин одной группы к дополнительному проводнику с нулевым (или пренебрежимо малым) сопротивлением. Такой дополнительный проводник физики обычно называют *шиной (bus)*, а способ моделирования перколяции с двумя группами вершин называют *геометрией шины (bus-bar geometry)* [3–9]. Способ моделирования, когда рассматривается протекание между всего двумя вершинами, в физике называют *двухточечной геометрией (two-point geometry)* [4, 5, 10, 11, 12]. Объединение всех простых цепей, соединяющих вершину из одной группы с вершиной из другой группы, принято называть *остовом (backbone)*. В качестве синонима термина *простая цепь* (*simple chain*) [13] иногда употребляется *несамопересекающийся путь* (*non-self-intersecting path*) [14] или *самоизбегающее блуждание* (*self-avoiding walk, SAW*) [15–18].

В некоторых публикациях отмечается, что следует различать *эффективный остов* (*effective backbone*) и *геометрический остов* (*geometric backbone*) [4, 19]. Под геометрическим остовом предлагается понимать данное выше определение остова как подграфа перколяционного кластера с двумя непересекающимися путями до противоположных краев решетки или эквивалентное ему. Однако не через все элементы геометрического остова будет протекать ток из-за наличия идеально сбалансированных связей, которые соответствуют мостам Уитстона. Соответственно, эффективным остовом называют только те элементы геометрического остова, через которые протекает ненулевой ток. Далее в этой статье рассматривается именно геометрический остов, называемый для краткости просто остовом.

Части перколяционного кластера, не входящие в остов, принято называть *висячими частями* (*dangling parts*) [20], хотя некоторые авторы употребляют другие названия, например, *запутывающие части* (*tangling parts*) [14]. В ряде публикаций, посвященных нахождению остова, приводится достаточно подробная классификация висячих частей: *висячие концы* (*dangling ends*), *висячие циклы* (*dangling loops*) и *висячие дуги* (*dangling arcs*) [3, 20].

При анализе графов, допускающих укладку на плоскости, могут быть использованы специальные алгоритмы, учитывающие планарность графа. К алгоритмам, специфическим для нахождения остова в плоском графе, относятся алгоритмы, предложенные Петером Грассбергером и Ренатом Ахунжановым.

Алгоритм Грассбергера. В 1992 году Грассбергером [14] был предложен алгоритм для нахождения остовов на квадратных решетках в перколяции узлов с открытыми граничными условиями. Такие решетки можно рассматривать как частный случай плоских графов. Грассбергер определяет остов как множество несамопересекающихся путей, соединяющих противоположные края решетки.

В алгоритме Грассбергера предлагается выполнять генерирование случайного кластера одновременно с его анализом. Однако тот факт, что случайный кластер генерируется прямо во время анализа, не является принципиальным, и предложенный Грассбергером алгоритм может быть применен и к заранее сгенерированному случайному графу на квадратной решетке. Генерирование случайного кластера выполняется в алгоритме Грассбергера подобно тому, как это делается в алгоритме Лиса [21–23], хотя в алгоритме Лиса генерирование кластера выполняется от одного начального узла, а в алгоритме Грассбергера генерирование занятого кластера выполняется от одного края решетки, что напоминает один из вариантов алгоритма, предложенных Джоном Хаммерсли. Хаммерсли в одном из вариантов изучения перколяции связей предлагал рассматривать смачивание от поверхности образца [24, с. 134–135].

П. Грассбергер рассматривает перколяцию узлов на квадратной решетке и использует термин *смоченные* (*wetted*) узлы в качестве синонима термина «занятые» узлы. Хотя термин «шина» не используется в статье Грассбергера, фактически для моделирования двух шин Грассбергер предлагает считать смоченными узлы, относящиеся к двум группам на противоположных краях решетки (нижнем крае на оси x и верхнем крае $y = L$), между которыми ищется перколяционный кластер и его остов. В статье приводится три версии программного кода: упрощенная, промежуточная и продвинутая. Хотя Грассбергер пишет в статье, что края $y = 0$ и $y = L$ содержат только занятые узлы [14, с. 5478], это верно только для упрощенной версии программного кода. Если анализировать программный код продвинутой версии, то фактически в коде предполагается, что узлы при $y = 0$ заняты случайным образом, а смоченными следует формально считать узлы, соответствующие $y = -1$. Информацию об узлах двумерной решетки Грассбергер предлагает хранить в виде двумерного целочисленного массива размером $L \times L$.

В упрощенной версии Грассбергер описывает более простой рекурсивный алгоритм, который просто проверяет, существует ли перколяционный кластер, давая наиболее левый соединяющий путь и висячие части кластера слева от него. Грассбергер выделяет четыре подпрограммы east, west, south, north, каждая соответствует одному шагу в одном из четырех соответствующих направлений. Каждая функция содержит внутри еще три вызова функций, соответствующих трем возможным направлениям дальнейшего движения (кроме обратного направления). Направления движения перебираются по часовой стрелке, например, функция north вызывает по порядку west, north, east. При помощи рекурсивных вызовов алгоритм генерирует разветвленный путь, который начинается при $x = y = 0$ и, по существу, следует левой части оболочки кластера, содержащего ось x. В упрощенной версии алгоритма программа помечает те узлы, для которых предпринималась попытка случайного заполнения. Хотя программа должна предполагать независимое заполнение узлов с некоторой вероятностью p, что соответствующий узел был занят, в упрощенной версии программы имеется неаккуратность, из-за которой с вероятностью p генерируется заполнение сразу трех соседей занятого узла. Упрощенная версия программы завершается при помощи

функции exit, вызываемой функцией north, когда достигнуто значение $y = L$, сообщая перед вызовом exit, что существует перколяционный кластер. Если $y = L$ никогда не достигается, то программа сообщает об отсутствии перколяционного кластера.

Грассбергер приводит фрагменты кода на языке Си, однако, по-видимому, оригинальная реализация программы Грассбергером была на языке Фортран, поскольку из фрагмента программы видно, что предполагается возвращение функцией rand() вещественного значения в диапазоне от 0 до 1, что соответствует реализации функции rand() в языке Фортран [25], в то время как в языке Си функция rand() возвращает целые значения в диапазоне от 0 до RAND_MAX [26].

Затем Грассбергер вносит две модификации в упрощенную версию алгоритма. Первая модификация заключается в том, чтобы пометку узлов при помощи двух значений $s[x][y] = 1$ или $s[x][y] = 0$ для проверенного/непроверенного узлов заменить пометкой при помощи трех значений: $s = 0$ (непроверенный), $s = q$ (занятый) и $s = INT\_MAX$ (пустой). Здесь q является любым положительным целым, меньшим, чем INT_MAX. Вторая модификация предусматривает наращивание глобального счетчика m каждый раз при входе в подпрограмму и уменьшение этого счетчика при выходе из подпрограммы. В результате, при помощи m считается количество узлов в остове (каждый рекурсивный вызов подпрограммы соответствует обработке одного узла). Грассбергер отмечает, что если эта программа останавливается, поскольку она достигла дальней стороны $y = L$, тогда все шаги в остове соответствуют вызовам подпрограмм, из которых еще не вышли, и значение m является как раз размером остова. Более точно, эта версия программы соответствует получению в m размера самого левого простого пути, принадлежащего остову. При модификации алгоритма Грассбергер также исправил отмеченную выше неаккуратность первоначальной версии, и здесь уже занятие узлов происходит независимо с вероятностью p. В результате описанных модификаций получается промежуточная версия алгоритма.

Далее Грассбергер предлагает еще одну модификацию алгоритма, чтобы считать также все остальные соединяющие пути. Для этого удаляется вызов exit в функции north и взамен наращивается q каждый раз, когда либо достигнут верхний край, либо если q больше, чем проверенное ненулевое $s[x][y]$. Также заменяется безусловное уменьшение m уменьшением по условию, что q не изменилось со времени входа в подпрограмму. Грассбергер пишет, что изменение q показывает, что после входа в подпрограмму либо был достигнут верхний край (в таком случае данный узел принадлежит остову), либо был достигнут какой-то узел, принадлежащий остову (в таком случае данный узел также принадлежит остову).

Грассбергер отмечает, что при отсутствии смены порядка перебора направлений движения будут некорректно определяться внутренние части остова. По этой причине Грассбергер предлагает запоминать направление движения: «вверх» или «вниз». При движении вверх Грассбергер предлагает брать сперва самые левые ветви (т.е. использовать перебор направлений по часовой стрелке), а при движении вниз брать сперва самые правые ветви (т.е. использовать перебор направлений против часовой стрелки).

Чтобы запоминать направление, Грассбергер предлагает кодировать его в четности/нечетности q, начиная с четного q (например, $q = 2$), наращивать q до следующего большего нечетного числа, когда достигается верхний край $y = L$, наращивать q до следующего большего четного числа, когда достигается нижний край $y = 0$ (в статье Грассбергера опечатка: $x = 0$), и наращивать q на две единицы, когда достигается любой другой узел остова [14, с. 5482]. Таким образом, Грассбергер предлагает кодировать движение вверх и перебор направлений по часовой стрелке четными q, а движение вниз и перебор направлений против часовой стрелки – нечетными q. Отметим, что термины «вверх» или «вниз» являются достаточно условными, поскольку при движении по цепочке занятых узлов направление может изменяться сложным образом, и после достижения верхнего края решетки возможен возврат опять к верхнему краю, а после достижения нижнего края решетки возможен возврат опять к нижнему краю.

Недостатки алгоритма Грассбергера и их частичное устранение. О некоторых недостатках алгоритма Грассбергера ранее кратко докладывалось на конференции [27], где отмечалось, что при поиске остова алгоритм корректно обнаруживает все висячие концы, но присоединяет некоторые висячие циклы и висячие дуги, однако конкретных примеров, когда алгоритм Грассбергера работает некорректно, в той публикации не было приведено. В [27] предлагался способ преодоления недостатков алгоритма Грассбергера за счет рассмотрения четырех ориентаций решетки с конкретным случайным заполнением: помимо исходной ориентации рассматривался поворот решетки на 180°, а также отражения относительно вертикальной и относительно горизонтальной осей. В [27] и для различных случайных заполнений решетки, и для различных ориентаций одного случайного заполнения использовался термин «конфигурация», что не совсем удачно. В данной статье различные ориентации одного случайного заполнения решетки называются ориентациями, а термин «конфигурация»

используется для различных случайных заполнений. После публикации [27] при сравнении результатов алгоритма Грассбергера с результатами алгоритма Ахунжанова было обнаружено, что даже при рассмотрении четырех ориентаций алгоритм Грассбергера обнаруживает не все висячие циклы. В данной публикации подробно рассмотрены примеры заполнения решеток, в которых алгоритм Грассбергера работает некорректно, в том числе и для четырех ориентаций.

Для исследования алгоритма Грассбергера нами была сделана реализация этого алгоритма на C++, максимально приближенная к фрагментам кода в тексте статьи Грассбергера, с исправлением небольших неточностей в коде, приводимом в статье [14], программа PureGrasberger.cpp [28]. Для удобства регулирования размеров решетки двумерный массив был реализован в программе PureGrasberger.cpp как вектор векторов. Для удобства анализа в программу была добавлена возможность сохранения сгенерированного случайного заполнения узлов в файле.

Затем была реализована возможность анализа тем же алгоритмом конкретного заполнения, читаемого из файла, с выводом обнаруженного остова в файл для детального анализа, программа GrassbergerFileBB.cpp [28].

В результате анализа был выявлен ряд конфигураций, в которых к остову присоединяются висячие циклы и висячие дуги. Характерной особенностью этих висячих частей является то, что алгоритм Грассбергера не обнаруживает висячие части «с одной стороны». А именно алгоритм Грассбергера определяет висячие циклы с левой стороны, но не определяет висячие циклы с правой стороны (при движении «вверх» и, наоборот, при движении «вниз»). На рисунке 1 показаны соответствующие конфигурации. Цветные ячейки с ненулевыми числами соответствуют заполненным узлам. Зеленые ячейки, заполненные единицами, помечают правильный остов; желтые ячейки, заполненные двойками, являются висячими циклами, которые определяет алгоритм Грассбергера; красные ячейки, заполненные тройками, являются висячими циклами, которые алгоритм Грассбергера ошибочно присоединяет к остову.

0	0	0	0	1	0	0	0	0
0	0	2	2	1	1	1	0	0
0	0	2	0	2	0	1	0	0
0	0	2	2	2	0	1	0	0
0	0	0	0	0	0	1	0	0
0	0	2	2	2	0	1	0	0
0	0	2	0	2	0	1	0	0
0	0	2	2	1	1	1	0	0
0	0	0	0	1	0	0	0	0

а

0	0	0	0	1	0	0	0	0
0	0	1	1	1	3	3	0	0
0	0	1	0	3	0	3	0	0
0	0	1	0	3	3	3	0	0
0	0	1	0	0	0	0	0	0
0	0	1	0	3	3	3	0	0
0	0	1	0	3	0	3	0	0
0	0	1	1	1	3	3	0	0
0	0	0	0	1	0	0	0	0

б

Рисунок 1 – Алгоритм Грассбергера: а) определяет висячие циклы слева; б) не определяет висячие циклы справа

Кроме этого, алгоритм Грассбергера обнаруживает висячие дуги с верхней стороны, но не обнаруживает висячие дуги с нижней стороны решетки. На рисунке 2 показана соответствующая конфигурация. Пометка цветами и числами аналогична рисунку 1.

0	2	0	2	0	0	1	0	2	0	2	0	0
0	2	0	2	0	1	1	0	2	0	2	0	0
0	2	0	2	0	1	0	0	2	0	2	0	0
0	2	0	2	0	1	0	0	2	2	2	0	0
0	2	0	2	0	1	0	0	0	0	0	0	0
0	2	0	2	0	1	0	0	0	3	3	3	0
0	2	0	2	0	1	1	1	0	3	0	3	0
0	2	2	2	0	0	0	1	0	3	0	3	0
0	0	0	0	0	0	0	1	0	3	0	3	0
0	0	3	3	3	0	0	1	0	3	0	3	0
0	0	3	0	3	0	0	1	0	3	0	3	0
0	0	3	0	3	0	1	1	0	3	0	3	0
0	0	3	0	3	0	1	0	0	3	0	3	0

Рисунок 2 – Алгоритм Грассбергера определяет висячие дуги сверху, но не определяет снизу

Для устранения проблемы висячих частей «с одной стороны» была реализована программа, которая помимо исходной решетки рассматривает еще три ориентации той же самой решетки: поворот решетки на 180 градусов, а также отражение относительно вертикальной и горизонтальной осей, программа GrassbergerFileBB4copy.cpp [28].

На рисунке 3 показаны конфигурации с четырьмя висячими циклами, которые алгоритму Грассбергера удается обнаружить лишь в одной из четырех ориентаций решетки (для каждого из четырех циклов своя ориентация).

0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
0	1	0	3	3	3	0	0	0	3	3	3	0	1	0
0	1	0	3	0	3	0	0	0	3	0	3	0	1	0
0	1	1	1	3	3	0	0	0	3	3	1	1	1	0
0	0	0	1	0	0	0	0	0	0	0	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	0	0	0	0	0	0	0	1	0	0	0
0	1	1	1	3	3	0	0	0	2	2	1	1	1	0
0	1	0	3	0	3	0	0	0	2	0	2	0	1	0
0	1	0	3	3	3	0	0	0	2	2	2	0	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1	0

Рисунок 3 – Алгоритм Грассбергера определяет каждый из висячих циклов лишь в одной из четырех ориентаций

Кроме этого, была сделана попытка распараллелить реализованный алгоритм с использованием технологии MPI. Рекурсивные алгоритмы, к которым относится алгоритм Грассбергера, плохо поддаются распараллеливанию, но было реализовано распределение четырех вышеупомянутых ориентаций одной и той же решетки между четырьмя MPI-процессами, программа GrassbergerFileBB4copyMPI.cpp [29]. Главный процесс занимается вводом решетки из файла, рассылкой введенной решетки на остальные процессы, обработкой основной ориентации решетки, сбором окончательного остова и выводом остова в файл. Остальные процессы получают от главного процесса массив с конфигурацией решетки, затем обрабатывают свою ориентацию решетки (поворот и отражения) и отправляют результат главному процессу.

Однако следует отметить, что существуют более сложные конфигурации висячих циклов, приведенные на рисунках 4 и 5, которые алгоритм Грассбергера не обнаруживает ни в одной из четырех ориентаций. Если узел остова, к которому присоединён висячий цикл, добавляется в остов до того, как произошёл заход в цикл, то висячий цикл тоже присоединяется к остову. Общей особенностью приведенных на рисунках 4 и 5 конфигураций является то, что в этих конфигурациях висячие циклы присоединены к циклам, принадлежащим остову. Хотя алгоритм Грассбергера делает периодическую смену направления поворота, но соответствующие принадлежащие остову циклы по алгоритму Грассбергера всегда проходятся так, что прикрепленный к ним висячий цикл всегда оказывается «с необнаруживаемой стороны». На рисунке 4 приведена конфигурация со вложенными висячими циклами, а на рисунке 5 приведена конфигурация с висячими циклами с краю. Также можно отметить, что конфигурации типа рисунка 5 могут реализовываться на меньших решетках, чем конфигурации типа рисунка 4.

0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	1	0	1	1	1	1	1	0	1	1	1	1	1	0	1	0
0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	1	0
0	1	0	1	0	3	3	1	1	1	3	3	0	1	0	1	0
0	1	0	1	0	3	0	3	0	3	0	3	0	1	0	1	0
0	1	0	1	0	3	3	3	0	3	3	3	0	1	0	1	0
0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0
0	1	0	1	0	3	3	3	0	3	3	3	0	1	0	1	0
0	1	0	1	0	3	0	3	0	3	0	3	0	1	0	1	0
0	1	0	1	0	3	3	1	1	1	3	3	0	1	0	1	0
0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	1	0
0	1	0	1	1	1	1	1	0	1	1	1	1	1	0	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Рисунок 4 – Алгоритм Грассбергера не определяет вложенные висячие циклы ни в одной из четырех ориентаций

0	1	0	0	0	0	0	0	0	0	0	1	0
0	1	0	3	3	3	0	3	3	3	0	1	0
0	1	0	3	0	3	0	3	0	3	0	1	0
0	1	0	3	3	1	1	1	3	3	0	1	0
0	1	0	0	0	1	0	1	0	0	0	1	0
0	1	1	1	1	1	0	1	1	1	1	1	0
0	0	0	0	0	1	0	1	0	0	0	0	0
0	1	1	1	1	1	0	1	1	1	1	1	0
0	1	0	0	0	1	0	1	0	0	0	1	0
0	1	0	3	3	1	1	1	3	3	0	1	0
0	1	0	3	0	3	0	3	0	3	0	1	0
0	1	0	3	3	3	0	3	3	3	0	1	0
0	1	0	0	0	0	0	0	0	0	0	1	0

Рисунок 5 – Алгоритм Грассбергера не определяет висячие циклы по краям ни в одной из четырех ориентаций

Алгоритм Ахунжанова. Для устранения проблем алгоритма Грассбергера, связанных с присоединением висячих циклов, Ренатом Камилевичем Ахунжановым был предложен рекурсивный алгоритм, корректно находящий остовы в плоских графах [15]. Отметим, что, хотя в теории перколяции обычно рассматриваются графы без петель, а в статье [15] говорится о применении алгоритма к простым плоским графам, предложенный Ахунжановым алгоритм может быть обобщен для применения и на плоских графах с петлями.

Что касается опубликованного в статье [15] описания алгоритма, то прежде всего следует отметить досадную опечатку в псевдокоде функции NG: в строке 5 должно браться не ребро E, а ребро E', т.е. вместо оператора

$$5{:}V_1' \leftarrow AdjacentVertex(E, V_1, G)$$

должен быть оператор

$$5{:}V_1' \leftarrow AdjacentVertex(E', V_1, G)$$

К сожалению, в статье [15] не были ясно обозначены многие идеи, касающиеся общей организации алгоритма Ахунжанова, в частности практически не пояснялся псевдокод функции WF, а большая часть пояснений псевдокода функции NG сводилась к описанию чисто технических вспомогательных функций для нахождения смежной вершины ($AdjacentVertex$) и следующего ребра, прикрепленного к вершине ($NextEdge$). В статье [18] была предпринята попытка дать более подробное описание именно идей, реализованных в алгоритме, но, к сожалению, некоторые фразы в статье [18] только запутывают описание алгоритма Ахунжанова.

В качестве одного из достоинств алгоритма авторы отмечают, что алгоритм посещает не все ребра графа. Однако по неясным причинам авторы меняют терминологию: если в [15] авторы говорят о «непосещенных ребрах» (unvisited edges), то в [18] авторы говорят о «непройденных ребрах» (untraversed edges), хотя термин «посещение» (visiting) продолжают использовать. По-видимому, авторы рассматривают термины «непосещенные» и «непройденные» как синонимы, поэтому неясно, в чем преимущество использования термина «непройденные».

Еще одним существенным моментом, затрудняющим восприятие объяснений в [18], является то, что, хотя авторы ссылаются на предыдущую работу [15], к сожалению, никак не упоминают выделяемые в [15] две подпрограммы: функцию WF и процедуру NG. Более того, в [18] авторы не только не упоминают названий подпрограмм из [15], но даже вообще не говорят о выделении двух подпрограмм, из-за чего очень сложно сопоставить объяснения в [18] с описанием алгоритма, приводимым в [15]. По-видимому, первый абзац из параграфа «A. Модифицированный алгоритм следования вдоль стен» (A. Modified wall follower algorithm) приблизительно соответствует описанию функции WF, а новый словесный вариант псевдокода, состоящий из шага 1 (Step 1) и шага 2 (Step 2), в этом параграфе приблизительно соответствует описанию процедуры NG.

При описании алгоритма в [15] и [18] авторы предпочитают называть простые цепи аббревиатурой SAW. К сожалению, в словесных формулировках, описывающих алгоритм, и в [15], и в [18] встречается ряд неточностей. В частности, в [15] авторы говорят, что ищут все простые пути между входной вершиной V_{in} и выходной вершиной V_{out}. На самом деле, алгоритм не выделяет все простые пути между V_{in} и V_{out}, а выделяет *множество ребер, принадлежащих всем простым путям* между V_{in} и V_{out}. В [18] встречается еще более странная фраза о поиске множества всех SAWs между двумя заданными вершинами (входом V_{in} и выходом V_{out}), принадлежащих внешнему периметру G. Выделение всех SAWs, принадлежащих внешнему периметру, алгоритмом не реализуется, возможно, авторы имели в виду просто *множество всех ребер, принадлежащих внешнему периметру* G. Самый первый вызов функции WF, в действительности, выделяет самую левую простую цепь, принадлежащую внешнему периметру G. Далее в [18], по-видимому, для пояснения функции WF говорится, что каждый дважды пройденный перекресток (вершина $\deg V > 1$) указывает на простой цикл. На самом деле, в реализации функции WF дважды пройденные перекрестки могут соответствовать не только простым циклам, но и любым *циклическим путям* (в простом цикле повторяются только начальная и конечная вершина, а в циклических путях могут повторяться любые вершины и ребра), которые являются висячими частями.

Перечисление всех неточностей в [15] и [18] заняло бы много места, поэтому мы отметим только еще один наиболее существенный, на наш взгляд, недостаток – запутанное объяснение, сколько раз проходятся какие из ребер. В [15] авторы говорят только о посещенных и непосещенных ребрах, не уточняя, что некоторые ребра посещаются более одного раза. В [18] авторы в оценке худшего случая говорят, что ребра остова проходятся дважды, хотя на самом деле ребра остова проходятся алгоритмом трижды.

Поэтому далее мы попытаемся дать наше описание алгоритма и прояснить вопрос, сколько раз проходятся какие из ребер.

Процедура NG организует многократные вызовы функции WF для выделения простых цепей, принадлежащих остову. Процедура NG при помощи рекурсии проходит все ребра, присоединенные к остову («зеленые» ребра в терминах [15]), и при наличии инцидентных очередной вершине ребер, которые могут принадлежать остову, вызывает для обработки этих ребер функцию WF. Основной идеей, обеспечивающей применение рекурсии, является то, что процедура NG проходит присоединенные ранее к остову простые цепи, проверяя возможность присоединения новых простых цепей. С данной идеей тесно связано упоминаемое, но не объясняемое в [15] добавление удаляемых впоследствии фантомных зеленых ребер. Добавление фантомных ребер используется для первоначального запуска процедуры NG, которая предполагает, что уже была определена некоторая часть остова, и присоединение новых простых цепей осуществляется при проходе ранее определенных частей остова.

Каждый вызов функции WF определяет «гроздь» висячих частей и/или новую простую цепь, принадлежащую остову. Возможно определение «грозди» висячих частей, присоединённых к вершине V, для которой вызывается функция WF, а если будет определена простая цепь, принадлежащая остову, то будут также определены и все имеющие место висячие части, присоединенные к этой простой цепи слева, если рассматривать проход по этой простой цепи от вершины V до другого конца простой цепи. Функция WF красит внешний периметр висячих частей в желтый цвет, а принадлежащую остову простую цепь – в зеленый цвет.

В целом, алгоритм Ахунжанова может проходить ребра обрабатываемого графа от 0 до 3 раз (рисунок 6 иллюстрирует количество проходов):

- 0 раз проходятся ребра, которые являются внутренними ребрами висячих частей, т.е. не принадлежат к внешнему периметру висячих частей (черные ребра на рисунке 6). Внешний периметр висячих частей – это внешний периметр графа, получаемого объединением всех висячих частей;
- 1 раз проходятся ребра, которые принадлежат внешнему периметру висячих частей, но не являются мостами. Проход по этим ребрам делается функцией WF и, согласно [15], красит их в желтый цвет (желтые ребра на рисунке 6);
- 2 раза проходятся ребра, которые принадлежат внешнему периметру висячих частей и являются мостами (коричневые ребра на рисунке 6). Оба прохода делаются функцией WF, один проход по мосту делается для захода в висячую часть, другой – для выхода. Эти проходы, согласно [15], дважды красят ребро в желтый цвет;
- 3 раза проходятся ребра, принадлежащие остову (зеленые ребра на рисунке 6). В том числе сначала два прохода делаются функцией WF; согласно [15], первый проход делается, когда еще не найден второй конец принадлежащей остову простой цепи, и красит ребро в желтый цвет, второй проход делается для пометки ребра как принадлежащего остову и красит ребро в зеленый цвет. Третий проход делается процедурой NG для проверки, существуют ли еще простые цепи, которые следует присоединить к остову.

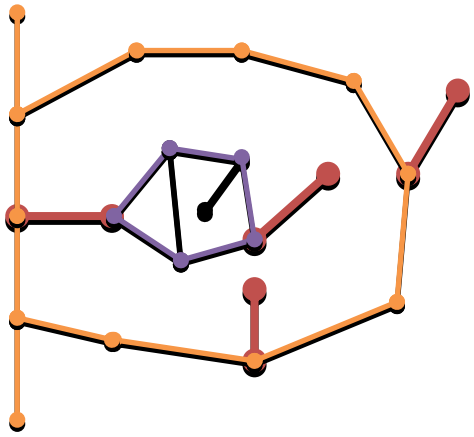


Рисунок 6 – Иллюстрация пояснения о количестве проходов по ребрам

Теоретическая оценка вычислительной сложности алгоритма Ахунжанова. В статье Ю.Ю. Тарасевича и др. [18] была проделана оценка вычислительной сложности алгоритма Ахунжанова для лучшего и для худшего случая. Авторы приводят оценку наилучшего случая сверху $O(\sqrt{N_v})$, рассматривая в качестве примера для наилучшего случая граф, показанный на рисунке 7.

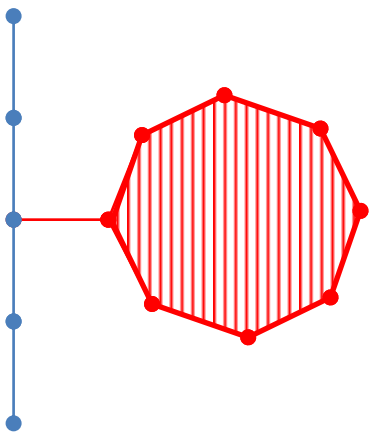


Рисунок 7 – Иллюстрация наилучшего случая в [18]

Однако не совсем ясно обоснование такой оценки. Авторы справедливо отмечают, что доля вершин и ребер, принадлежащих внешнему периметру графа, существенно зависит от структуры графа. Однако затем авторы делают вызывающее вопросы утверждение, что число вершин едва ли превышает $\sqrt{N_V}$, за исключением некоторых специально сконструированных графов. Следует отметить, что приводимый авторами граф является как раз специально сконструированным, а отнюдь не типичным для перколяционных задач. Если учесть, что авторы в своей статье рассматривают случайное осаждение стержней, концы и пересечения которых авторы считают вершинами графа, то при неравномерном осаждении стержней в некоторых областях моделирования может быть получена сколь угодно более высокая концентрация вершин. Рассматривая, например, случаи, когда огромное количество стержней попало внутрь красной заштрихованной области, можно получить графы с гораздо меньшей долей вершин, принадлежащих периметру, чем $\sqrt{N_V}$, например, $\sqrt[3]{N_V}$. Если считать такие графы лучшим случаем, то для них получается, соответственно, оценка сложности $O(\sqrt[3]{N_V})$, что существенно лучше предлагаемой авторами [18] оценки $O(\sqrt{N_V})$.

Можно предположить, что авторы имели в виду оценку лучшего случая «в среднем», хотя явно о том, что рассматривается усредненная оценка, авторы не пишут. Косвенным указанием на оценку лучшего случая в среднем является фраза о «специально сконструированных графах», поскольку эта фраза имеет обычно смысл лишь в контексте усреднения и означает в таком контексте, что случаи, называемые «специально сконструированными», имеют очень малую вероятность и не вносят заметного вклада в среднее значение. Авторы статьи справедливо отмечают, что наименьшая доля стержней и, соответственно, узлов принадлежит остову на пороге перколяции. Таким образом, если говорить о лучшем случае «в среднем», то можно предполагать, что авторы делали оценку лучшего случая «в среднем» на пороге перколяции. Однако при такой трактовке получается, что авторы [18] совсем забыли о том, что и кратчайший путь, который на рисунке 7 будет остовом, и внешний периметр (перколяционного кластера), который необходимо обойти на рисунке 7, на пороге перколяции имеют «в среднем» фрактальный характер.

В литературе можно найти соответствующие фрактальные размерности. Например, определенные с высокой точностью значения фрактальной размерности кратчайшего (минимального) пути можно найти в [32], где приводится значение $d_{min} = 1.13077(2)$, и в [33], где приводится значение $d_{SP} = 1.130 \pm 0.005$. Менее точное значение фрактальной размерности минимального пути $D_{min} = 1.13$ приводится в известной монографии Штауффера и Аарони [1, с. 52]. Точное значение фрактальной размерности внешнего периметра перколяционного кластера $D_e = \frac{4}{3}$ было получено для двумерной перколяции на основе модели кулоновского газа Салье и Дюплантье в 1987 году [34]. Недавние численные эксперименты показывают прекрасное согласие с точным значением данной размерности [35, 36].

Среднее количество вершин в графе N_V пропорционально квадрату размера моделируемой области L:

$$N_V \sim L^2, \tag{1}$$

откуда

$$L \sim N_V^{\frac{1}{2}}. \tag{2}$$

В то же время среднее количество вершин графа N_{min} во фрактальной структуре, имеющей размерность D_{min}, пропорционально соответствующей степени размера моделируемой области:

$$N_{min} \sim L^{D_{min}}. \tag{3}$$

Подставляя (2) в (3), получаем

$$N_{min} \sim \left(N_V^{\frac{1}{2}}\right)^{D_{min}} = N_V^{\frac{D_{min}}{2}}. \tag{4}$$

Используя значение $D_{min} = 1.13$ в формуле (4), получаем $N_{min} \sim N_V^{0,565}$, т.е. если сложность алгоритма пропорциональна числу вершин в кратчайшем пути N_{min}, то можно дать оценку сложности $O\left(N_V^{0,565}\right)$. Таким образом, при рассмотрении лучшего случая в среднем, даже если рассматривать обработку только кратчайшего пути с учетом его фрактальности, оценка сложности $O\left(N_V^{0,565}\right)$ оказывается заметно выше предлагаемой авторами [18] оценки $O(\sqrt{N_V})$.

Для среднего количества вершин графа во внешнем периметре N_e можно получить аналогично формуле (4) оценку

$$N_e \sim N_V^{\frac{D_e}{2}}. \tag{5}$$

Подставляя значение $D_e = \frac{4}{3}$ в формуле (5), получаем $N_e \sim N_V^{\frac{2}{3}}$, т.е. если сложность алгоритма пропорциональна числу вершин во внешнем периметре N_e, то можно дать оценку сложности

$O\left(N_V^{\frac{4}{3}}\right)$. Таким образом, если при рассмотрении лучшего случая в среднем рассматривать обработку вершин во внешнем периметре, то с учетом его фрактальности, оценка сложности $O\left(N_V^{\frac{4}{3}}\right)$ оказывается еще выше, чем предлагаемая авторами [18] оценка $O(\sqrt{N_V})$.

Особенности реализации алгоритма Ахунжанова. В приложении к статье Р.К. Ахунжанова и др. [15] на странице одного из соавторов статьи Ю.Ю. Тарасевича приводится реализация алгоритма на языке Python для плоских двумерных графов, образованных случайным осаждением стержней (файл BBSearcher.py) [29]. Разработчиком данной реализации является А.В. Есеркепов. Следует отметить некоторые существенные моменты, отличающие реализацию Есеркепова от описания алгоритма в статье [15].

Прежде всего, отметим, что в статье [15] процедура NG и функция WF имеют по четыре аргумента: $NG(E, V_1, V_2, G)$ и $WF(E_1, V, E_2, G)$, в то время как у Есеркепова [29] четвертый аргумент, соответствующий графу G, отсутствует. Вместо этого Есеркепов использует глобальные переменные, хранящие информацию о графе. Такой подход вполне разумен, чтобы уменьшить количество накапливаемых в стеке аргументов при рекурсивных вызовах процедуры NG, когда предполагается обработка всего лишь одного графа.

Еще одно существенное отличие программы Есеркепова заключается в том, что в ней была использована другая конфигурация добавочных ребер, отличающаяся от описанной в статье [15]. Согласно статье, добавляется два зеленых ребра, соединяющих вершины V_{in} и V_{out}. Одно из этих ребер используется в качестве начального ребра E_0, с которого начинается выполнение процедуры NG. Однако в программе Есеркепова добавляется только одно зеленое ребро между вершинами V_{in} и V_{out}, а еще одно зеленое ребро добавляется между вершиной V_{in} и еще одной, не описанной в статье добавочной вершиной V_0. Именно это ребро (V_0, V_{in}) используется в качестве начального ребра E_0 в программе Есеркепова при первом вызове процедуры $NG(E_0, V_{in}, V_{out})$. На рисунке 8 показано фактическое положение дополнительных ребер, соответствующее программе Есеркепова.

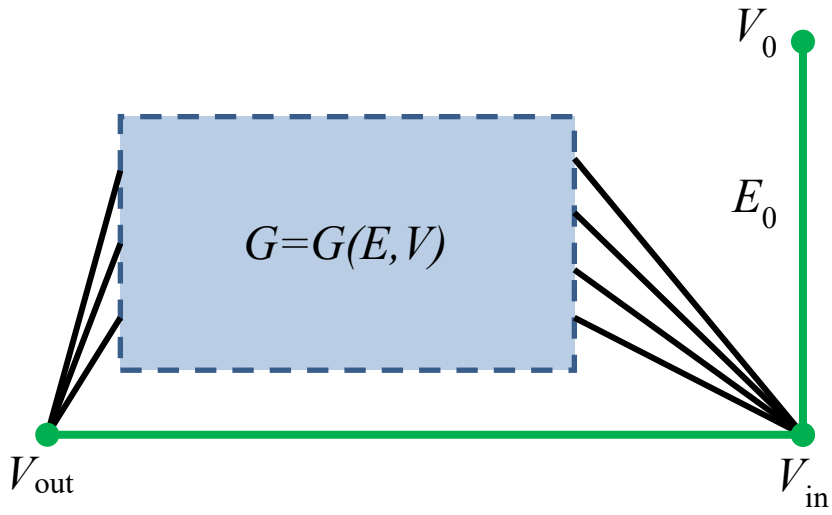


Рисунок 8 – Иллюстрация фактического расположения дополнительных ребер в программе Есеркепова

Такой вариант конфигурации дополнительных ребер для начального запуска процедуры NG тоже работает, поскольку для алгоритма Ахунжанова существенно только, чтобы существовал путь из зеленых ребер через вершину V_{in} до вершины V_{out}, а в конечном счете дополнительные ребра все равно удаляются. Однако следует более подробно остановиться на том, почему в реализации Есеркепова изменено расположение одного из дополнительных зеленых ребер. Дело в том, что соединение вершин V_{in} и V_{out} двумя различными ребрами превращает граф G в мультиграф, в котором есть кратные ребра между некоторыми парами вершин. Алгоритм Ахунжанова может с успехом обрабатывать плоские мультиграфы, но в реализации Есеркепова имеется еще одна не описанная в статье особенность, из-за которой была изменена конфигурация дополнительных ребер. Дело в том, что в реализации Есеркепова при каждом обращении к вершине выполняется сортировка прикрепленных к ней ребер против часовой стрелки, так чтобы рассматриваемое ребро оказалось первым в списке. С целью сортировки у Есеркепова рёбра рассматриваются как отрезки,

соединяющие вершины с определенными координатами на плоскости, но при таком подходе кратные ребра невозможно различить.

На самом деле, нет необходимости выполнять сортировку ребер при каждом обращении к вершине, гораздо эффективнее формировать список ребер, инцидентных вершине, сразу в порядке против часовой стрелки и рассматривать сформированный список ребер как циклический, что и было сделано нами при реализации алгоритма Ахунжанова в виде программы на C++ [30]. При этом нет необходимости рассматривать ребра как отрезки, соединяющие две точки на плоскости, а можно рассматривать соединение вершин и произвольными непрерывными линиями, что является необходимым для плоских мультиграфов. Отметим, что наша реализация алгоритма допускает обработку плоских мультиграфов и два зеленых добавочных ребра добавляются именно так, как описывается в статье Ахунжанова.

Экспериментальное сравнение алгоритмов. Было проделано экспериментальное сравнение алгоритмов по времени выполнения и доле отличий в остове на случайных решетках. Аналогично статье Грассбергера брались решетки с размерами $L = 5, 7, 10, 14, 20, 28, \ldots, 1280, 1792, 2560, 3584, 5120$. По сравнению со статьей Грассбергера максимальная сторона решетки была увеличена в 4 раза – с $L = 1280$ до $L = 5120$, соответственно, число узлов на решетке было увеличено в 16 раз. Для решетки каждого размера рассматривалось по 200 случайных реализаций. Поскольку в статье Грассбергера оценивались остовы на решетках, сгенерированные случайным смачиванием от низа решетки, то для объективности оценки неточностей в определении остова алгоритмом Грассбергера нами при тестировании всех алгоритмов рассматривались только случайные смачивания снизу решетки, сгенерированные с помощью программы PureGrasberger.cpp [28].

На рисунке 9 показана средняя доля отличий остова, получаемого алгоритмом Грассбергера, от остова, получаемого алгоритмом Ахунжанова, на решетках с размерами от $L = 80$ до $L = 5120$. Погрешность средней доли отличий показана на графике вертикальными штрихами. На всех случайных решетках такого размера обнаруживались отличия в остовах. На случайных решетках от $L = 5$ до $L = 56$ встречались экземпляры решеток без отличий. При $80 \leq L \leq 160$ было замечено некоторое возрастание среднего значения, затем определенной тенденции не было обнаружено.

Следует отметить, что, хотя средняя доля отличий не превышает 20 %, на некоторых случайных решетках остовы отличались более чем в 2 раза; максимально обнаруженное отличие на случайных решетках составляло 2,78 раза, т.е. 178 %. В файле InputGraph56_00167.txt [31] приводится пример случайной решетки с $L = 56$, на которой обнаружено отличие более чем в два раза. Остов, выделенный для этой решетки алгоритмом Грассбергера, приводится в файле BackboneGrasFile BB56_00167.txt, а остов, выделенный для этой решетки алгоритмом Ахунжанова, приводится в файле BackboneAkhunNoSortTailNoBus56_00167.txt [31]. Обнаруженный остов в обоих выходных файлах показан единицами.

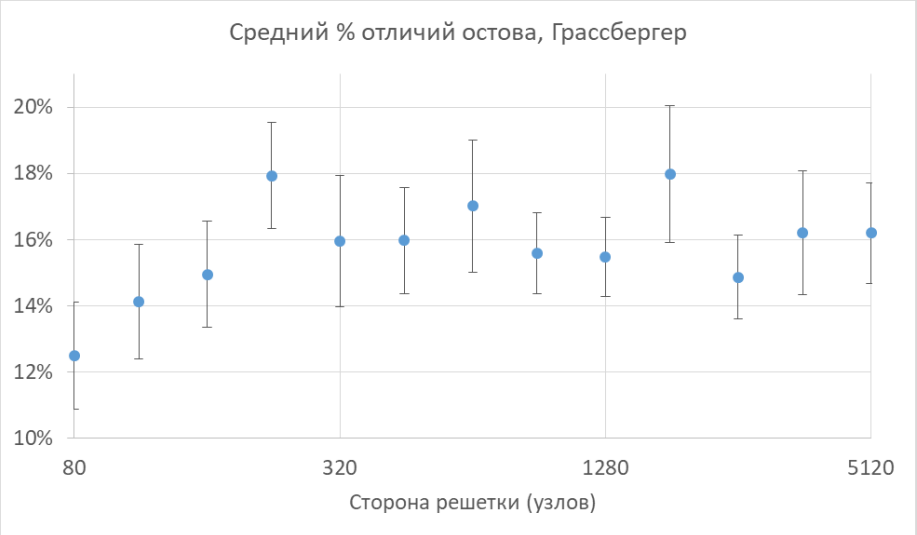


Рисунок 9 – Средний процент отличий остова, полученного алгоритмом Грассбергера, от остова, полученного алгоритмом Ахунжанова для разных размеров решеток

На рисунке 10 показана средняя доля отличий остова, получаемого применением алгоритма Грассбергера к четырем ориентациям решетки (отражения и повороты), от остова, получаемого алгоритмом Ахунжанова, на решетках с размерами от $L = 1792$ до $L = 5120$. Погрешность средней доли отличий показана на графике вертикальными штрихами. На всех случайных решетках такого размера обнаруживались отличия в остовах. На случайных решетках от $L = 40$ до $L = 1280$ встречались экземпляры решеток как с отличиями, так и без отличий. На случайных решетках от $L = 5$ до $L = 28$ отличий не встречалось. Заметна тенденция к увеличению отличия с увеличением размера решетки.

Следует отметить, что, хотя среднее отличие остова не превышало 0,05 %, на некоторых случайных решетках небольшого размера отличие составляло более 1%; максимальное обнаруженное отличие составляло 3,7 %. В файле InputGraph40_00178.txt [31] приводится пример случайной решетки с $L = 40$, на которой обнаружено отличие в 3,7 %. Остов, выделенный для этой решетки применением алгоритма Грассбергера с четырьмя ориентациями, приводится в файле Backbone Gras4copy40_00178.txt, а остов, выделенный для этой решетки алгоритмом Ахунжанова, приводится в файле BackboneAkhunNoSortTailNoBus40_00178.txt [31]. Обнаруженный остов в обоих выходных файлах показан единицами. Можно заметить, что отличие в 3,7 % было обнаружено в случае конфигурации типа, показанной на рисунке 5 (циклы по краям).

Для четырех версий программы был произведен замер времени, затрачиваемого в среднем на обработку одной решетки со стороной L узлов. Усреднение времени работы проводилось по 200 различным случайным заполнениям решетки. На рисунке 11 показаны графики зависимости среднего времени обработки решетки для четырех версий программы: алгоритм Грассбергера; алгоритм Грассбергера с четырьмя ориентациями; алгоритм Грассбергера с распараллеливанием на основе MPI по четырем ориентациям; алгоритм Ахунжанова. Стандартная погрешность для среднего времени обработки решетки не превышает размеров маркеров. На графиках показаны размеры решеток начиная с $L = 20$, поскольку на меньших размерах решеток от $L = 5$ до $L = 14$ погрешность в замерах времени оказывалась слишком большой.

Следует отметить, что с точки зрения точности определения остова алгоритм Грассбергера с четырьмя ориентациями и алгоритм Грассбергера с распараллеливанием на основе MPI по четырем ориентациям давали одинаковый результат, но на больш*и*х решетках распараллеленный алгоритм оказывался быстрее в 1,4 раза. Самым быстрым из четырех алгоритмов является алгоритм Грассбергера, который дает самую большую неточность в определении остова. На малых решетках от $L = 5$ до $L = 80$ самым медленным оказывался алгоритм Грассбергера с распараллеливанием на основе MPI по четырем ориентациям, а на больших решетках от $L = 112$ до $L = 5120$ самым медленным оказывался алгоритм Ахунжанова. Для алгоритма Грассбергера с четырьмя ориентациями на решетках от $L = 5$ до $L = 640$ распараллеленный на основе MPI алгоритм оказывался медленнее, чем нераспараллеленный, а на решетках от $L = 896$ до $L = 5120$ распараллеленный алгоритм оказывался быстрее. Более медленная работа на малых решетках объясняется тем, что доля времени затрачиваемого на взаимодействие программы со средой выполнения MPI превышает время, затрачиваемое на вычисления, а на больших решетках это время относительно невелико.

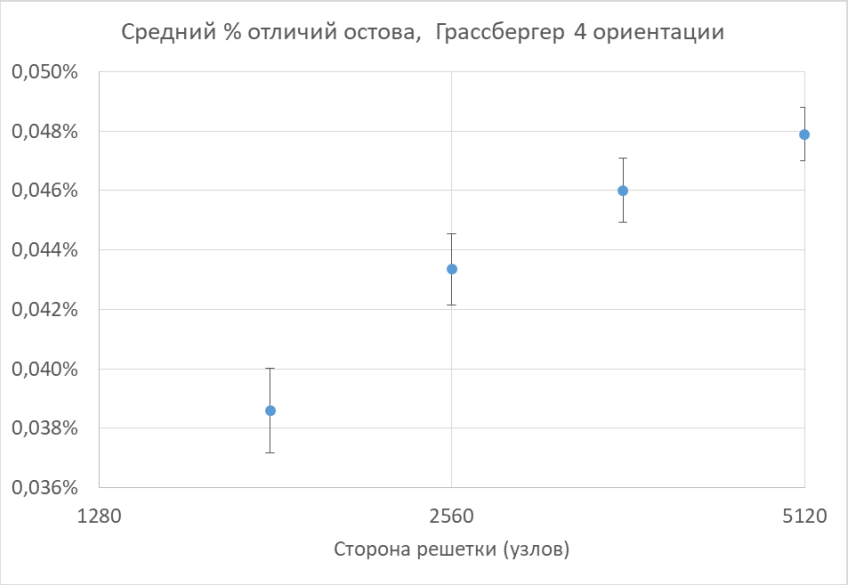


Рисунок 10 – Средний процент отличий остова, полученного алгоритмом Грассбергера с использованием четырех ориентаций, от остова, полученного алгоритмом Ахунжанова для разных размеров решеток

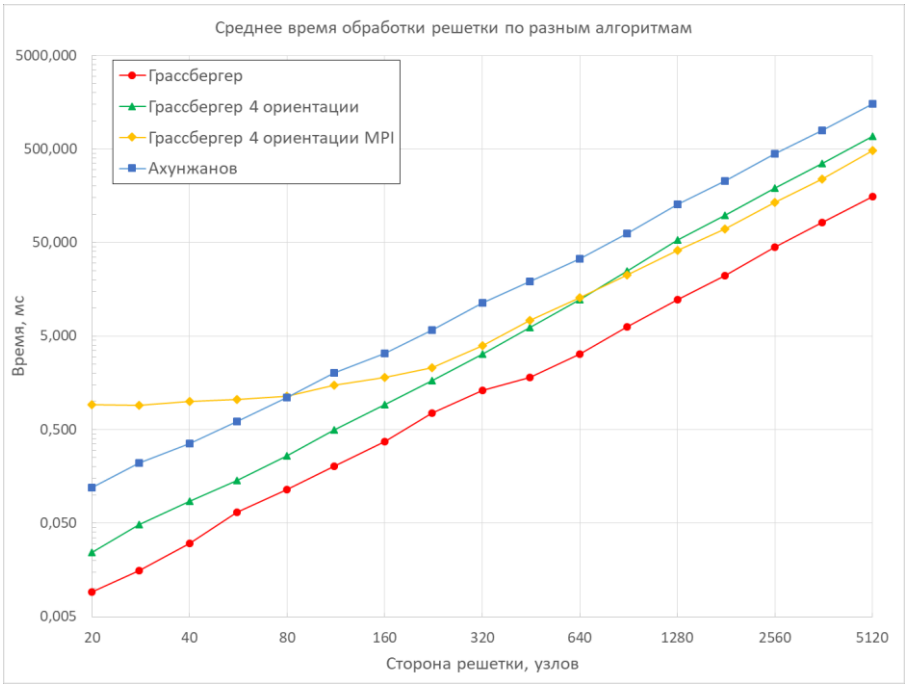


Рисунок 11 – График среднего времени обработки решетки для разных алгоритмов в зависимости от стороны решетки

Для тестирования всех программ использовался компьютер со следующими характеристиками:

- модель процессора – Intel (R) Core (TM) i3-10110U CPU;
- частота – 2,10 GHz;
- количество ядер – 2;
- количество логических процессоров – 4;
- количество оперативной памяти – 8 Гб;
- операционная система Windows 10.

Заключение. Был проделан сравнительный анализ алгоритма Грассбергера и алгоритма Ахунжанова для нахождения остова на квадратных решетках. Были выявлены недостатки алгоритма Грассбергера, связанные с неточным определением остова, и рассмотрен способ частичного устранения недостатков за счет анализа четырех ориентаций решетки.

Для алгоритма Ахунжанова был сделан более подробный теоретический анализ трудоемкости алгоритма, а также уточнены детали, связанные с количеством проходов по ребрам. Также было проделано численное сравнение результатов работы алгоритмов Ахунжанова и Грассбергера и сравнение времени их работы на решетках размером до 5120 х 5120 узлов.

Авторы благодарят А.В. Есеркепова за предоставленные исходные коды на языке Python. Также авторы благодарят Р.К. Ахунжанова и Ю.Ю. Тарасевича за стимулирующие обсуждения.

Библиографический список

1. Stauffer D. Introduction to Percolation Theory / D. Stauffer, A. Aharony. – London : Taylor and Francis, 1992.
2. Complex Media and Percolation Theory / eds: M. Sahimi, A. G. Hunt. – New York : Springer, 2021. 439 p.
3. Гордеев, И. И. Нахождение проводящего остова в двумерной решетке методом заливки / И. И. Гордеев, С. С. Овчаренко, А. А. Сизова // Прикаспийский журнал: управление и высокие технологии. – 2020. – № 1 (49). – С. 94–111. – DOI: 10.21672/2074-1707.2020.49.4.094-111.
4. Tarasevich, Yu. Yu. Identification of current-carrying part of a random resistor network: electrical approaches vs. graph theory algorithms / Yu. Yu. Tarasevich, A. S. Burmistrov, V. A. Goltseva, I. I. Gordeev, V. I. Serbin, A. A. Sizova, I. V. Vodolazskaya, D. A. Zholobov // Journal of Physics: Conference Series. – 2018. – Vol. 955. – P. 012021. – DOI: 10.1088/1742-6596/955/1/012021.
5. Redner, S. Fractal and Multifractal Scaling of Electrical, Conduction in Random Resistor Networks / S. Redner // Mathematics of Complexity and Dynamical Systems. – 2011. – P. 446–462.
6. Ioselevich, A. S. Percolation with excluded small clusters and Coulomb blockade in a granular system / A. S. Ioselevich, D. S Lyubshin // Письма в Журнал экспериментальной и теоретической физики. – 2009. – Т. 90, № 10. – С. 746–752.

7. Deng, Y. Magnetic and backbone exponents of the percolation and Ising models in three dimensions / Y. Deng, H. W. J. Blöte // Physical Review E. – 2004. – Vol. 70. – P. 046106.

8. Jesper, L. J. A transfer matrix for the backbone exponent of two-dimensional percolation / L. J. Jesper, Z.-J. Paul // Journal of Physics A: Mathematical and General. – 2002. – Vol. 35 (9). – P. 2131–2144.

9. Grassberger, P. Conductivity exponent and backbone dimension in 2-d percolation / P. Grassberger // Physica A: Statistical Mechanics and its Applications. – 1999. – Vol. 262. – P. 251–263.

10. Large, M. J. Finite-size scaling in silver nanowire films: design considerations for practical devices / M. J. Large, M. Cann, S. P. Ogilvie, A. A. K. King, I. Jurewicz, A. B. Dalton // Nanoscale. – 2016. – Vol. 8. – P. 13701–13707.

11. Large, M. J. Selective mechanical transfer deposition of Langmuir grapheme films for high-performance silver nanowire hybrid electrodes / M. J. Large, S. P. Ogilvie, S. Alomairy, T. Vöckerodt, D. Myles, M. Cann, H. Chan, I. Jurewicz, A. King, A. B. Dalton // Langmuir. – 2017. – Vol. 33, № 43. – P. 12038–12045.

12. Kovacs, G. J. Effect of the substrate on the insulator–metal transition of vanadium dioxide films / G. J. Kovacs, D. Bürger, I. Skorupa, H. Reuther, R. Heller, H. Schmidt // Journal of Applied Physics. – 2011. – Vol. 109. – P. 063708.

13. Fleischner, H. Algorithms in Graph Theory. TU Wien, Algorithms and Complexity Group / H. Fleischner. – March 11, 2016. – Режим доступа: https://www.dbai.tuwien.ac.at/staff/kronegger/misc/AlgorithmsInGraph Theory_Script.pdf, свободный. – Заглавие с экрана. – Яз. рус. (дата обращения: 11.08.2022).

14. Grassberger, P. Spreading and backbone dimensions of 2D percolation / P. Grassberger // Journal of Physics A: Mathematical and General. – 1992. – Vol. 25, № 21. – P. 5475–5484.

15. Akhunzhanov, R. K. Identification of a current-carrying subset of a percolation cluster using a modified wall follower algorithm / R. K. Akhunzhanov, A. V. Eserkepov, Y. Y. Tarasevich // Journal of Physics: Conference Series. – 2021. – Vol. 1740. – P. 012008.

16. Hong, D. C., Stanley H. E. Exact enumeration approach to fractal properties of the percolation backbone and 1/σ expansion / D. C. Hong, H. E. Stanley // Journal of Physics A: Mathematical and General. – 1983. – Vol. 16. – P. L475–L481.

17. Hong, D. C. Cumulant renormalisation group and its application to the incipient infinite cluster in percolation / D. C. Hong, H. E. Stanley // Journal of Physics A: Mathematical and General. – 1983. – Vol. 16. – P. L525–L529.

18. Tarasevich, Yu. Yu. Random nanowire networks: Identification of a current-carrying subset of wires using a modified wall follower algorithm / Yu. Yu. Tarasevich, R. K. Akhunzhanov, A. V. Eserkepov, M. V. Ulyanov // Physical Review E. – 2021. – Vol. 103. – P. 062145.

19. Roux, S. A new algorithm to extract the backbone in a random resistor network / S. Roux, A. Hansen // Journal of Physics A: Mathematical and General. – 1987. – Vol. 20. – P. L1281–L1285.

20. Yin, W.-G. Rapid algorithm for identifying backbones in the two-dimensional percolation model / W.-G Yin, R. Tao // International Journal of Modern Physics C. – 2003. – Vol. 14, № 10. – P. 1427–1437.

21. Leath, P. L. Cluster size and boundary distribution near percolation threshold / P. L. Leath // Physical review B. – 1976. – Vol. 14. – P. 5046.

22. Stauffer, D. Percolation Simulation: Large Lattices, Varying Dimensions / D. Stauffer, N. Jan // Annual Reviews of Computational Physics VIII. – Singapore, New Jersey, London, Hong Kong : World Scientific, 2001. – P. 287–300.

23. Гордеев, И. И. Сравнение различных описаний и реализаций алгоритма Лиса для перколяционных задач / И. И. Гордеев, А. А. Письменская // Научный форум: Технические и физико-математические науки : сб. ст. по материалам LIII Междунар. науч.-практ. конф. – Москва : МЦНО, 2022. – № 3 (53). – С. 49–55.

24. Hammersley, J. M. Monte Carlo Methods / J. M. Hammersley, D. C. Handscomb. – London : Me-thuen & Co Ltd., 1975.

25. RAND – Real pseudo-random number // The GNU Fortran Compiler. – Режим доступа: https://gcc.gnu.org/onlinedocs/gfortran/RAND.html, свободный. – Заглавие с экрана. – Яз. англ. (дата обращения: 30.08.2021).

26. Kernighan, B. W. The C programming language / B. W. Kernighan, D. M. Ritchie. – London : Prentice-Hall International (UK) Limited, 1988.

27. Гордеев, И. И. Анализ недостатков алгоритма Грассбергера для поиска перколяционного остова на квадратной решетке и возможности его распараллеливания / И. И. Гордеев, Н. С. Саенко // Актуальные проблемы информационно-телекоммуникационных технологий и математического моделирования в современной науке и промышленности : материалы I Междунар. науч.-практ. конф. молодых учёных, Комсомольск-на-Амуре, 20–25 марта 2021 г. / редкол.: А. Л. Григорьева (отв. ред.), Я. Ю. Григорьев, И. А. Трещев. – Комсомольск-на-Амуре : ФГБОУ ВО «КнАГУ», 2021. С. 12–14. – DOI: 10.17084/978-5-7765-1488-3-2021-12.

28. G2ii2g. Grassberger-backbone2 // GitHub. – Режим доступа: https://github.com/G2ii2g/Grassberger-backbone2, свободный. – Заглавие с экрана. – Яз. англ. (дата обращения: 28.07.2022).

29. Eserkepov, A. V. BBSearcher.py / A. V. Eserkepov // ResearchGate. – Режим доступа: https://www.researchgate.net/publication/349366739_BBSearcherpy/link/602cc40a4585158939ad6dd5/download, свободный. – Заглавие с экрана. – Яз. англ. (дата обращения: 07.05.2021).

30. G2ii2g. Akhunzhanov-backbone // GitHub. – Режим доступа: https://github.com/G2ii2g/Akhunzhanov-backbone, свободный. – Заглавие с экрана. – Яз. англ. (дата обращения: 08.08.2022).

31. G2ii2g. Grassberger-backbone2/tests // GitHub. – Режим доступа: https://github.com/G2ii2g/Grassberger-backbone2/tree/main/tests, свободный. – Заглавие с экрана. – Яз. англ. (дата обращения: 08.08.2022).

32. Zhou, Z. Shortest-path fractal dimension for percolation in two and three dimensions / Z. Zhou, J. Yang, Y. Deng, R. Ziff // Physical Review E. – 2012. – Vol. 86. – P. 061101.

33. Schrenk, K. J. Percolation with long-range correlated disorder / K. J. Schrenk, N. Posé, J. J. Kranz, L. V. M. van Kessenich, N. A. M. Araújo, H. J. Herrmann // Physical Review E. – 2013– . Vol. 88. – P. 052102.

34. Saleur, H. Exact Determination of the Percolation Hull Exponent in Two Dimensions / H. Saleur, B. Duplantier // Physical Review Letters. – 1987. – Vol. 58. – P. 2325–2328.

35. Xu, X. Geometric structure of percolation clusters / X. Xu, J. Wang, Z. Zhou, T. M. Garoni, Y. Deng // Physical Review E. – 2014. – Vol. 89. – P. 012120.

36. d'Auriac, J.-C. A. Statistics of percolating clusters in a model of photosynthetic bacteria / J.-C. A. d'Auriac, F. Iglói // Physical Review E. – 2021. – Vol. 103. – P. 052103.

References

1. Stauffer, D., Aharony, A. *Introduction to Percolation Theory*. London, Taylor and Francis, 1992.

2. Sahimi, M., Hunt, A. G. (eds). *Complex Media and Percolation Theory*. New York : Springer, 2021. 439 p.

3. Gordeev, I. I., Ovcharenro, S. S., Sizova, A. A. Nakhozhdenie provodyashchego ostova v dvumernoy reshetke metodom zalivki [The determination of the conducting backbone in a two-dimensional lattice by the flooding method]. *Prikaspiyskiy zhurnal: upravlenie i vysokie tekhnologii* [Caspian Journal: Control and High Technologies], 2020, no. 1 (49), pp. 94–111. DOI: 10.21672/2074-1707.2020.49.4.094-111.

4. Tarasevich, Yu. Yu., Burmistrov A. S., Goltseva V. A., Gordeev I. I., Serbin V. I., Sizova A. A., Vodolazskaya I. V., Zholobov D. A. Identification of current-carrying part of a random resistor network: electrical approaches vs. graph theory algorithms. *Journal of Physics: Conference Series*, 2018, Vol. 955, p. 012021. DOI: 10.1088/1742-6596/955/1/012021.

5. Redner, S. Fractal and Multifractal Scaling of Electrical, Conduction in Random Resistor Networks. *Mathematics of Complexity and Dynamical Systems*, 2011, pp. 446–462.

6. Ioselevich, A. S., Lyubshin, D. S. Percolation with excluded small clusters and Coulomb blockade in a granular system. *Pisma v Zhurnal eksperimentalnoy b teoreticheskoy fiziki* [Journal of Experimental and Theoretical Physics Letters], 2009, vol. 90, no. 10, pp. 746–752.

7. Deng, Y., Blöte, H. W. J. Magnetic and backbone exponents of the percolation and Ising models in three dimensions. *Physical Review E*. 2004. Vol. 70. 046106.

8. Jesper, L. J., Paul, Z.-J. A transfer matrix for the backbone exponent of two-dimensional percolation. *Journal of Physics A: Mathematical and General*, 2002, vol. 35 (9), pp. 2131–2144.

9. Grassberger, P. Conductivity exponent and backbone dimension in 2-d percolation. *Physica A: Statistical Mechanics and its Applications*, 1999, vol. 262, pp. 251–263.

10. Large, M. J., Cann, M., Ogilvie, S. P., King, A. A. K., Jurewicz, I., Dalton, A. B. Finite-size scaling in silver nanowire films: design considerations for practical devices. *Nanoscale*, 2016, vol. 8, pp. 13701–13707.

11. Large, M. J., Ogilvie, S. P., Alomairy, S., Vöckerodt, T., Myles, D., Cann, M., Chan, H., Jurewicz, I., King, A., Dalton, A. B. Selective mechanical transfer deposition of Langmuir grapheme films for high-performance silver nanowire hybrid electrodes. *Langmuir*, 2017, vol. 33, no. 43, pp. 12038–12045.

12. Kovacs, G. J., Bürger, D., Skorupa, I., Reuther, H., Heller, R., Schmidt, H. Effect of the substrate on the insulator–metal transition of vanadium dioxide films. *Journal of Applied Physics*, 2011, vol. 109, p. 063708.

13. Fleischner, H. *Algorithms in Graph Theory*. TU Wien, Algorithms and Complexity Group, March 11, 2016. Available at: https://www.dbai.tuwien.ac.at/staff/kronegger/misc/AlgorithmsInGraphTheory_Script.pdf (accessed 11.08.2022).

14. Grassberger, P. Spreading and backbone dimensions of 2D percolation. *Journal of Physics A: Mathematical and General*, 1992, vol. 25. no. 21. pp. 5475–5484.

15. Akhunzhanov, R. K., Eserkepov, A. V., Tarasevich, Y. Y. Identification of a current-carrying subset of a percolation cluster using a modified wall follower algorithm. *Journal of Physics: Conference Series*, 2021, vol. 1740, p. 012008.

16. Hong, D. C., Stanley, H. E. Exact enumeration approach to fractal properties of the percolation backbone and 1/σ expansion. *Journal of Physics A: Mathematical and General*, 1983, vol. 16, pp. L475–L481.

17. Hong, D. C., Stanley, H. E. Cumulant renormalisation group and its application to the incipient infinite cluster in percolation. *Journal of Physics A: Mathematical and General*, 1983, vol. 16, pp. L525–L529.

18. Tarasevich, Yu. Yu., Akhunzhanov, R. K., Eserkepov, A. V., Ulyanov, M. V. Random nanowire networks: Identification of a current-carrying subset of wires using a modified wall follower algorithm. *Physical Review E*., 2021, vol. 103. 062145.

19. Roux, S., Hansen, A. A new algorithm to extract the backbone in a random resistor network. *Journal of Physics A: Mathematical and General*, 1987, vol. 20, pp. L1281–L1285.

20. Yin, W.-G., Tao, R. Rapid algorithm for identifying backbones in the two-dimensional percolation model. *International Journal of Modern Physics C*., 2003, vol. 14, no. 10, pp. 1427–1437.

21. Leath, P. L. Cluster size and boundary distribution near percolation threshold. *Physical review B*., 1976, vol. 14, p. 5046.

22. Stauffer, D., Jan, N. Percolation Simulation: Large Lattices, Varying Dimensions. *Annual Reviews of Computational Physics VIII*. Singapore, New Jersey, London, Hong Kong : World Scientific, 2001, pp. 287–300.

23. Gordeev, I. I., Pismenskaya, A. A. Sravnenie razlichnykh opisaniy i realizatsiy algoritma Lisa dlya perkolyatsionnykh zadach [Comparision of different descriptions and implementations of the Leath algorithm for percolation problems]. *Nauchnyy forum: Tekhnicheskie i fiziko-matematicheskie nauki : sbornik statey po materialam LIII*

mezhdunarodnoy nauchno-prakticheskoy konferentsii [Scientific forum: Technical and physical and mathematical sciences : a collection of articles based on the materials of the LIII International scientific-practical conference]. Moscow, ICSE LLC, 2022, no. 3 (53), pp. 49–55.

24. Hammersley, J. M., Handscomb, D. C. *Monte Carlo Methods*. London, Me-thuen & Co Ltd., 1975.

25. *RAND – Real pseudo-random number*. The GNU Fortran Compiler. Available at: https://gcc.gnu.org/onlinedocs/gfortran/RAND.html (accessed 30.08.2021).

26. Kernighan, B. W., Ritchie, D. M. *The C programming language*. London, Prentice-Hall International (UK) Limited, 1988.

27. Gordeev, I. I., Saenko, N. S. Analiz nedostatkov algoritma Grassbergera dlya poiska perkolyacionnogo ostova na kvadratnoj reshetke i vozmozhnosti ego rasparallelivaniya [Analysis of disadvantages of Grassberger's algorithm finding percolating backbone on square lattice and the possibility of its parallelization]. *Aktualnye problemy informatsionno-telekommunikatsionnykh tekhnologiy i matematicheskogo modelirovaniya v sovremennoy nauke i promyshlennosti: materialy I Mezhdunarodnoy nauchno-prakticheskoy konferentsii molodykh uchenykh, Komsomolsk-na-Amure, 20–25 marta 2021* [Actual problems of information and telecommunication technologies and mathematical modeling in modern science and industry: materials of the I International scientific-practical conference of young scientists, Komsomolsk-on-Amur, March 20–25, 2021]. Komsomolsk-on-Amur, FGBOU VO « KnAGU », 2021, pp. 12−14. DOI: 10.17084/978-5-7765-1488-3-2021-12.

28. G2ii2g. *Grassberger-backbone2*. GitHub. Available at: https://github.com/G2ii2g/Grassberger-backbone2 (accessed 28.07.2022).

29. Eserkepov, A. V. *BBSearcher.py*. ResearchGate. Available at: https://www.researchgate.net/publication /349366739_BBSearcherpy/link/602cc40a4585158939ad6dd5/download (accessed 07.05.2021).

30. G2ii2g. *Akhunzhanov-backbone*. GitHub. Available at: https://github.com/G2ii2g/Akhunzhanov-backbone (accessed 08.08.2022).

31. G2ii2g. *Grassberger-backbone2/tests*. GitHub. Available at: https://github.com/G2ii2g/Grassberger-backbone2/tree/main/tests (accessed 08.08.2022).

32. Zhou, Z., Yang, J., Deng, Y., Ziff, R. Shortest-path fractal dimension for percolation in two and three dimensions. *Physical Review E*., 2012, vol. 86, p. 061101.

33. Schrenk, K. J., Posé, N., Kranz, J. J., van Kessenich, L. V. M., Araújo, N. A. M., Herrmann, H. J. Percolation with long-range correlated disorder. *Physical review E*., 2013, vol. 88, p. 052102.

34. Saleur, H., Duplantier, B. Exact Determination of the Percolation Hull Exponent in Two Dimensions. *Physical Review Letters*, 1987, vol. 58, pp. 2325–2328.

35. Xu, X., Wang, J., Zhou, Z., Garoni, T. M., Deng, Y. Geometric structure of percolation clusters. *Physical Review E*., 2014, vol. 89, p. 012120.

36. d'Auriac, J.-C. A., Iglói, F. Statistics of percolating clusters in a model of photosynthetic bacteria. *Physical Review E*. 2021. vol. 103, p. 052103.