УДК 519.688 DOI: 10.26102/2310-6018/2019.25.2.030

Э.С.Симонян, О.А.Медведева, С.Н.Медведев

СОЗДАНИЕ ЛАБИРИНТА С НЕСКОЛЬКИМИ ПРОХОДАМИ, ПОИСК ВСЕХ ПУТЕЙ В НЕМ И ИХ РЕДАКТИРОВАНИЕ

Воронежский государственный университет,
Воронеж, Россия

В статье рассмотрены задачи построения лабиринта с несколькими проходами, нахождения всех проходов и возможности их редактирования. Вначале рассмотрены два алгоритма генерации лабиринтов, дающие наиболее разнообразные лабиринты на выходе, выявлены их достоинства и недостатки, выбран один наиболее эффективный и подходящий к данной задаче. Проблемой всех алгоритмов построения лабиринтов, и двух рассмотренных в том числе, является невозможность создания лабиринта с определенным количеством проходов, поэтому предложена модификация алгоритма Уилсона. Проблемой всех известных алгоритмов нахождения проходов в лабиринте является то, что ни один из них не находит абсолютно все проходы. Поэтому для нахождения всех путей разработана модификация муравьиного алгоритма. Для редактирования лабиринта разработаны два алгоритма: на основе длины путей и на основе схожести путей. Проведено исследование с целью выявить зависимость количества путей в лабиринте от начальных параметров. Для модификации муравьиного алгоритма проведен вычислительный эксперимент для выявления оптимального количества муравьев при различных входных параметрах. Кроме того, проведена оценка эффективности алгоритмов редактирования лабиринта с различными входными данными.

Ключевые слова: генерация лабиринта, муравьиный алгоритм, алгоритм Уилсона, вычислительный эксперимент.

1. Введение

В настоящее время значимость лабиринтов в развлекательной сфере возрастает. Задача их прохождения встречается во многих проектах различных жанров. По мере увеличения мощностей вычислительной техники, размерности и скорость построения лабиринтов также возрастают. Однако если лабиринты небольшой размерности можно нарисовать вручную, то сложность генерации лабиринтов большой размерности намного возрастает. Существуют многочисленные алгоритмы генерации лабиринтов [1-3], со своими особенностями, ограничениями, достоинствами и недостатками. Также есть алгоритмы поиска путей в лабиринте. Однако существуют практические задачи, которые не решаются стандартными алгоритмами. Одна из таких задач будет представлена в статье, и для нее потребуется конструирование новых методов и подходов к решению такой задачи.

Лабиринт можно представить, используя, так называемый, «лабиринт с толстыми стенками» – двумерный массив, каждый элемент которого может быть либо стенкой лабиринта (значение 0), либо клеткой (значение 1) – свободным пространством, по которому можно передвигаться.

На Рисунке 1 представлена визуализация данного представления. Матрицу лабиринта, состоящую из 0 и 1, можно представить визуально набором пикселей, где пиксель черного цвета – стенка, а пиксель белого цвета – клетка лабиринта. Несложно сделать вывод, что при такой реализации размерность лабиринта обязательно должна быть нечетным числом. Действительно, у всех стенок одна из координат всегда является нечетной. Например, в левом верхнем углу стенка имеет координаты (1,1) стенка ниже имеет координаты (2,1), стенка правее – (1,2). Дальнейшая генерация лабиринта заключается в вырезании стен, т.е. в замене стен на клетки. Для простоты положим вход в лабиринт в левом верхнем углу, а выход из него в нижнем правом, т.е. движение начинается с точки (2,2) и заканчивается в точке $(N-1, N-1)$.

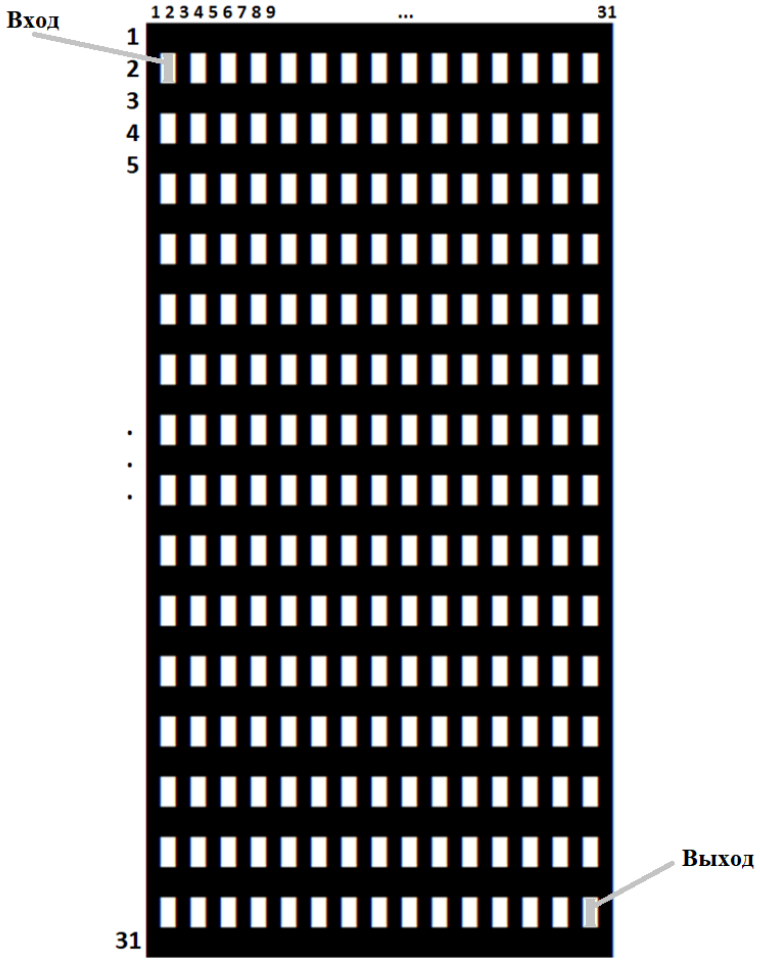

Рисунок 1 - Представление лабиринта

2. Постановка задачи

Необходимо построить лабиринт заданного размера с определенным количеством путей его прохождения (иначе говоря, проходами), которые удовлетворяет некоторым условиям. Такими условиями могут быть длины и количество проходов, их схожесть и т.п.

На данный момент не существует алгоритмов, которые генерируют лабиринты с заданным числом проходов. Поэтому можно выделить следующие основные этапы реализации поставленной задачи:

1. Построить лабиринт с одним входом, одним выходом и некоторым числом проходов, которое, строго говоря, до окончания построения неизвестно.
2. Найти все возможные проходы сгенерированного лабиринта.
3. Реализовать алгоритмы удаления неподходящих и ненужных путей с целью уменьшения общего числа проходов.

3. Материалы и методы

3.1. Генерация лабиринта

Известно большое количество алгоритмов генерации лабиринтов [3]. Каждый из них обладает своими достоинствами и недостатками, при этом наибольший интерес вызывают два алгоритма: алгоритм Олдоса-Бродера и алгоритм Уилсона. Два данных алгоритма особенны тем, что все возможные варианты лабиринтов генерируются с равной вероятностью, а также тем, что алгоритмы не имеют предпочтения в выборе направления, а работают одинаково по всем четырем. Обе особенности позволяют алгоритмам Уилсона и Олдоса-Бродера строить непохожие друг на друга лабиринты. Рассмотрим кратко каждый из них.

Идея алгоритма Олдоса-Бродера заключается в построении лабиринта из случайно выбранных непосещенных клеток. Соответственно, клетки могут быть двух видов – непосещенные и добавленные в лабиринт.

Алгоритм 1 [3]**.** Алгоритм Олдоса-Бродера

0. Инициализировать лабиринт, выбрав в нем случайную клетку.
1. Выбрать любую соседнюю (для текущей клетки) клетку. Соседних клеток может быть 2, 3 или 4, в зависимости от расположения клетки.
2. Если соседняя клетка не является частью лабиринта – добавить ее в лабиринт, т.е. убрать стенку, соединяющую две клетки. Иначе переход к шагу 3.
3. Если не осталось непосещенных клеток – останов. Иначе переход к шагу 1.

Алгоритм использует только соседние клетки. Это означает, что если на нулевом шаге выбрать, например, клетку $(2,2)$, то соседней для нее будет, например, клетка $(4,2)$. Согласно алгоритму, клетка $(4,2)$ будет добавлена в

лабиринт, после чего алгоритм снова перейдет на шаг 1, на котором будет снова выбирать соседнюю клетку, но уже для текущей, то есть для клетки (4,2). Но вследствие того, что в алгоритме не предусмотрено использование памяти для посещенных и непосещенных клеток, соседней для клетки (4,2) может снова оказаться клетка (2,2). Такого рода «блуждание» по уже посещенным клеткам и является главным недостатком алгоритма, и при увеличении размерностей лабиринта время его построения стремится к бесконечности. Приведем пример: на поле размером 101×101 остались две непосещенные клетки. Вероятность случайным образом попасть хотя бы в одну из них близка к нулю.

Алгоритм Уилсона устраняет этот минус – все непосещенные клетки заносятся в список. Также данный алгоритм удаляет циклы, если таковые образуются. Циклом называется ситуация, когда в лабиринте присутствует замкнутый путь из некоторой клетки в саму себя.

Алгоритм 2 [3]**.** Алгоритм Уилсона

1. Занести все клетки лабиринта в список непосещенных клеток.
2. Выбрать любую начальную клетку, добавить ее в лабиринт, убрать из списка непосещенных.
3. Выбрать случайным образом клетку из списка непосещенных.
4. Выбрать случайную соседнюю клетку для текущей клетки.
5. Если соседняя клетка является частью лабиринта – добавить пройденные на текущей итерации клетки в лабиринт, убрать стенки между ними, убрать из списка непосещенных клеток.
   5.1. Если образовался цикл – удалить его, т.е. снова занести в список непосещенных те клетки, которые образовали цикл, продолжив движение от клетки, с которой цикл начинался.
   5.2. Иначе перейти в соседнюю клетку и продолжать выполнять шаг 4 до тех пор, пока не выполнится шаг 5.
6. Если список непосещенных клеток пуст – останов. Иначе перейти к шагу 3.

Необходимо пояснить метод удаления циклов. Предположим, что движение начато из клетки (3,3). Затем алгоритм Уилсона найдет множество A непосещенных клеток (их число зависит только от выбора случайных клеток на шагах 3 и 4), последняя из которых на очередном шаге 4 алгоритма привела снова к клетке (3,3). Это значит, что необходимо сделать все клетки из найденного набора A снова непосещенными. Далее необходимо продолжить выполнение алгоритма из точки (3,3). Стоит отметить, что данные действия не гарантируют, что при дальнейшей работе алгоритма из точки (3,3) он не найдет множество клеток A (возможно,

состоящее из тех же самых клеток), которое снова приведет к образованию цикла.

Можно сделать вывод, что алгоритм Уилсона работает намного эффективнее, но требует дополнительных затрат памяти для хранения списка непосещенных клеток. Также стоит отметить и наличие у данного алгоритма средства удаления циклов, которое замедляет время его работы.

Алгоритм Уилсона, благодаря своим особенностям, является наиболее подходящим для поставленной задачи. Но его базовый вариант генерирует лабиринт только с одним проходом. Поэтому в статье предлагается модификация алгоритма Уилсона.

Такую модификацию позволяет сделать еще одна особенность: для алгоритма не важна форма поля лабиринта, поэтому можно генерировать лабиринт частями. Иными словами, можно задать любые ограничения на индексы двумерного массива. Например, строить лабиринт алгоритмом Уилсона в случае, когда индексы удовлетворяют некоторым ограничениям, например, $5 < i < 21$, а $j > i/2$, где i отвечает за строки лабиринта, а j – за столбцы лабиринта.

Примечательно, что в лабиринте, построенном в ограниченной неразрывной области, также будет проход из любой клетки в любую. Поэтому было введено понятие цикличной части лабиринта – это часть лабиринта, которая образует замкнутое пространство (Рисунок 2 б). Соответственно, кроме самой цикличной части, обозначенной как 1, поле разбивается еще на две части – часть внутри цикла (2) и вне его (3). Цикличную часть можно задать, например, следующим образом:

$$A = \begin{cases} \left[\frac{N}{8}\right] < i < \left[N - \frac{N}{8}\right], \\ \left[\frac{N}{8}\right] < j < \left[N - \frac{N}{8}\right]. \end{cases}$$

$$B = \begin{cases} \left[\frac{N}{2} - 3\right] < i < \left[\frac{N}{2} + 3\right], \\ \left[\frac{N}{2} - 3\right] < j < \left[\frac{N}{2} + 3\right]. \end{cases}$$

И взять область $A \backslash B$.

Алгоритм 3. Модифицированный алгоритм Уилсона

1. В начальном лабиринте (рис 2 а) выделить цикличную часть.
2. Базовым алгоритмом Уилсона построить лабиринт в цикличной части (Рисунок 2 б).

3. Базовым алгоритмом Уилсона построить лабиринты внутри цикла (Рисунок 2 в) и вне его (Рисунок 2 г).
4. Сделать вырез на границе цикличной и внешней некоторое заранее заданное число проходов (Рисунок 2 д).

Поясним шаг 4. Принцип вырезания может быть следующим: если граница области представляет собой прямоугольник, то можно сделать некоторое число вырезов k на двух смежных сторонах этого прямоугольника, после чего сделать на двух других сторонах k диаметрально противоположных вырезов. Таким образом, будет всегда четное число вырезов (примечание: при двух вырезах в лабиринте будет ровно один проход).

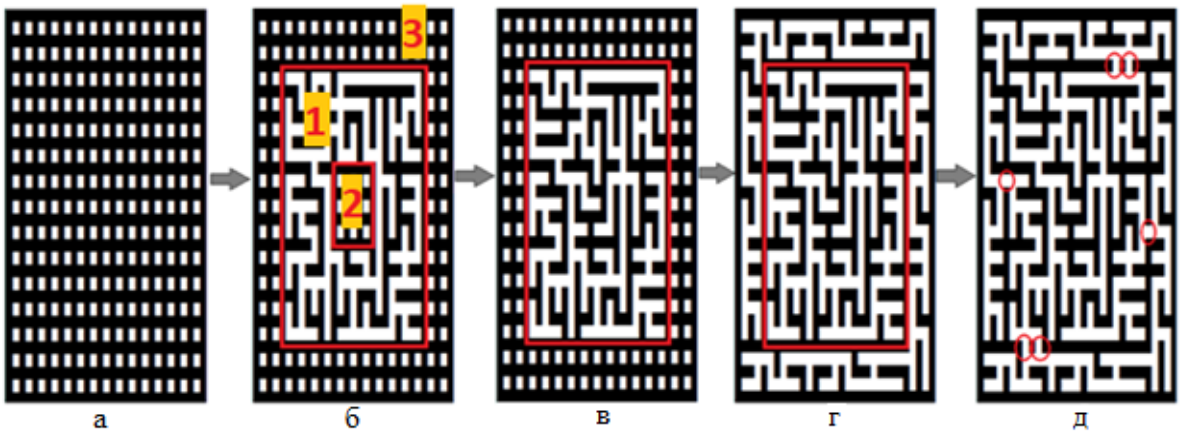

Рисунок 2 - Модифицированный алгоритм Уилсона

3.2. Нахождение всех путей в лабиринте

Для поиска путей в лабиринте существует множество алгоритмов [3], однако все они имеют свою специализацию. Например, алгоритм Collision Solver находит только самые короткие пути, а Chain Algorithm в качестве ответа дает только первый найденный путь. Также существует несколько алгоритмов для нахождения наибольшего числа проходов (в идеале всех). Есть алгоритмы, которые могут как пропустить некоторое число проходов, так и добавить проходы, которые по окончании работы алгоритма не будут являться решением лабиринта. Остальная часть алгоритмов имеет другой принцип работы: вместо поиска путей алгоритм находит только множество клеток, которые не ведут в тупик. А это значит, что для получения множества проходов необходимо добавить еще один алгоритм, который будет из множества клеток получать множество проходов. Иными словами, для получения множества проходов необходимо второй раз пройти лабиринт. Следовательно, среди всех алгоритмов нет такого, который находил бы все существующие пути. Поэтому разработка таких алгоритмов является актуально задачей. В данной статье взят за основу и

модифицирован муравьиный алгоритм. Для начала рассмотрим базовый муравьиный алгоритм [4].

Для решения любой задачи муравьиным алгоритмом необходимо представить ее в виде набора вершин и ребер. Если спроецировать такое представление на лабиринт, то вершина – это любая клетка лабиринта, а ребро – проход между двумя любыми клетками.

Общие этапы муравьиного алгоритма:

1. Создать муравьев на стартовой точке, которая зависит от ограничений, накладываемых на задачу.
2. Задать начальное число феромонов.
3. Поиск пути происходит на основе вероятности перехода из вершины i в вершину j, которая определяется по следующей формуле:

$$p_{ij}^{t} = \frac{(\tau_{i\,j}^{t})^{\alpha}(\frac{1}{c_{i\,j}})^{\beta}}{\sum_{l\in J}(\tau_{i\,l}^{t})^{\alpha}(\frac{1}{c_{i\,l}})^{\beta}}, \quad (1)$$

где t – номер итерации,
$\tau_{i\,j}^{t}$ – уровень феромонов на ребре (i,j) на итерации t,
$c_{i\,j}$ – расстояние между $i\ u\ j$,
α, β – некоторые константы,
J – множество вершин, доступных на итерации t *из вершины* i.

4. Пока не закончатся муравьи, повторять шаг 3.
5. Обновить феромоны в соответствии с формулой:

$$\tau_{ij}^{t+1} = (1-\rho)\tau_{ij}^{t} + \sum_{k\epsilon A_{ij}} \frac{Q}{L_k}, \quad (2)$$

где t – номер итерации,
$\tau_{i\,j}^{t}$ – уровень феромонов на ребре (i,j) на итерации t,
ρ – интенсивность испарения феромонов,
L_k – длина пути, пройденного k-м муравьем,
A_{ij} – множество муравьев, проходивших по (i,j),
Q – общее количество феромонов у муравья,
$\frac{Q}{L_k}$ – количество феромонов, откладываемых k-м муравьем
на ребро(i,j).

6. Если не выполнено условие останова, то повторять шаг 3.

Отличие разработанного алгоритма нахождения всех путей лабиринта от базового заключается в следующем:

1. Расстояния между вершинами не должны влиять на вероятность перехода, поэтому положим, что длина пути, пройденного

муравьем, не важна и формула (1) вероятности перехода из вершины i в j на итерации t изменяется следующим образом:

$$p_{ij}^t = \frac{(\tau_{ij}^t)^\alpha}{\sum_{l \in J}(\tau_{il}^t)^\alpha}, \quad (3)$$

где t – номер итерации,
τ_{ij}^t – уровень феромонов на ребре (i,j) на итерации t,
α, β – некоторые константы,
J – множество вершин, доступных на итерации t из вершины i.

2. Муравьи откладывают фиксированное количество феромонов, не зависящее от количества посещенных клеток. Тогда формула (2) обновления феромонов упрощается:

$$\tau_{ij}^{t+1} = (1-\rho)\tau_{ij}^t + L, \quad (4)$$

где t – номер итерации,
τ_{ij}^t – уровень феромонов на ребре (i,j) на итерации t,
ρ – интенсивность испарения феромонов,
$L = const$ – количество феромонов, которые откладывает муравей.

Введем также в алгоритм дополнительные параметры на основе поставленной задачи:

1. Матрица навигации – матрица, каждый элемент которой обозначает число дальнейших возможных направлений из текущей клетки. Очевидно, что матрица будет состоять только из чисел 0 (стена), 1 (тупик или вход/выход), 2, 3 и 4. Размерность матрицы совпадает с размерностью лабиринта.
2. В лабиринт вводятся пометки, т.е. места возможных изменений в нем. Самый простой для реализации вариант – оставлять пометки в виде порядкового номера муравья в клетке, в которую он переходит после каждой развилки. Таким образом, в каждой пометке будет содержаться массив из порядковых номеров муравьев, которые посетили данную клетку.

Таким образом, модифицированный муравьиный алгоритм выглядит следующим образом:

Алгоритм 4. Модифицированный муравьиный алгоритм

0. Построить матрицу навигации P_Crit.
1. Задать число муравьев N_Ant.
2. Задать одинаковое количество феромонов для каждого направления на каждой развилке.

3. $k = 1$.
4. Запустить k-го муравья.
5. Внести в текущую клетку пометку.
6. Если $P\_Crit_{ij} = 1$ и клетка является входом в лабиринт – перейти в единственном возможном направлении до следующей развилки. Перейти к шагу 5.
7. Если $P\_Crit_{ij} = 1$ и клетка является выходом из лабиринта – k-й путь найден. Проверить путь на уникальность и перейти к шагу 18.
8. Если 6 и 7 не выполнены и $P\_Crit_{ij} = 1$ – найден тупик. Вернуться обратно на предыдущую развилку, обнулить феромоны в текущее направление. Убрать пометки, оставленные в пройденном направлении. Перейти к шагу 5.
9. Если $P\_Crit_{ij} = 2$ и клетка является входом в лабиринт – перейти к шагу 13.
10. Если $P\_Crit_{ij} = 2$ и клетка является выходом из лабиринта – k-й путь найден. Проверить путь на уникальность и перейти к шагу 18.
11. Если 9 и 10 не выполнены, то продолжить движение в единственно возможном направлении до ближайшей развилки.
Перейти к шагу 5.
12. Если $P\_Crit_{ij} = 3$ или $P\_Crit_{ij} = 4$ – перейти к шагу 13.
13. Вычислить вероятность p_{ij}^t по формуле (2).
14. Если $p_{i,j}^t = 0$ для всех направлений, то вернуться на предыдущую развилку. Обнулить феромон для текущего направления. Убрать пометки, оставленные в текущем направлении. Перейти к шагу 5. Иначе перейти к шагу 15.
15. Если найдена развилка, в которой муравей уже был – найден цикл. Вернуться обратно на предыдущую развилку и обнулить «фиктивные» феромоны на данное направление. Убрать пометки, оставленные в цикле. Перейти к шагу 5. Иначе перейти к шагу 16.
16. Осуществить вероятностный переход в следующую клетку.
17. Обновить феромоны по формуле (4). Перейти к шагу 5.
18. Если $k = N\_Ant$ – останов. Иначе $k = k + 1$; перейти к шагу 4.

Необходимо пояснить 15-й шаг. Он подразумевает введение «фиктивных» феромонов, т.е. феромонов, которые действуют только на текущего муравья, и которым возвращается исходное значение после запуска следующего муравья. Это необходимо в том случае, если один

муравей обнулит феромоны в одном из направлений вследствие цикла. Тогда остальные муравьи из-за обнуленного феромона могут потерять один или несколько путей в лабиринте. «Фиктивные» феромоны решают данную проблему, однако жертвуют временем поиска пути.

Также можно оптимизировать шаги, в которых оставляются пометки. Например, можно не оставлять пометки в первой и последней развилке. Более того, можно не оставлять пометки на развилке или развилках, следующих за начальной точкой (даже если из начальной клетки есть два возможных направления), а также не оставлять пометки на развилках, предшествующих выходу из лабиринта. Причина заключается в том, что они не будут задействованы в алгоритмах удаления путей, иными словами, в этих клетках никогда не появятся стены, так как если бы они появились, лабиринт перестал бы быть решаемым.

3.3. Редактирование лабиринта

Под редактированием понимается удаление некоторых путей, т.е. добавление в лабиринт стен. Для этой цели предлагается использовать оставленные муравьями пометки. Для реализации данной задачи в статье разработаны два алгоритма редактирования: алгоритм удаления по длине и алгоритм удаления схожих путей.

Алгоритм 5. Алгоритм удаления по длине

1. Задать интервал длин путей, которые необходимо оставить.
2. Отыскать одну или несколько пометок, содержащих как можно больше путей, длины которых не входят в заданный интервал, и как можно меньше путей, входящих в заданный интервал.
3. Поставить в найденных пометках стены.

Для следующего алгоритма необходимо ввести параметр k – данный параметр задает процент совпадения путей. Например, $k = 80$ означает, что алгоритм найдет несколько множеств путей, которые покоординатно совпадают не менее чем на 80% и оставит из каждого множества только один путь, остальные же пути необходимо закрыть. Принцип отбора пути из каждого множества можно задать произвольно. Например, можно оставлять только самый короткий путь в каждом множестве.

Алгоритм 6. Алгоритм удаления схожих путей.

1. Задать параметр k.
2. Найти множества путей, совпадающих не менее чем на k%.
3. Выбрать из каждого множества один путь, остальные пути поместить в список путей, которые необходимо закрыть.

4. Найти пометку с максимальным количеством путей, которые необходимо удалить, и минимальным числом путей, которые необходимо оставить.
5. Поставить в найденную пометку стену.

Замечание 1: в шаге 3 алгоритма №6 пути, которые не были оставлены, исключаются из дальнейшего поиска совпадений. Например, если путь 1 совпадает с путями 2 и 3, и из них был оставлен только путь 2, то пути 1 и 3 помещаются во множество путей, подлежащих удалению, а следующая итерация поиска начинается с пути 4 и не рассматривает путь 2.

Замечание 2: для выполнения шага 2 алгоритма №6 можно использовать следующие действия:

1) Для первого пути вычисляем величину

$$c = \frac{s_i}{l} \cdot 100\%, \qquad i = 2..N \tag{5}$$

где s_i – количество координат, совпадающих у первого и i-го путей,

$l = \max(l_1, l_i)$ – максимум из длин двух путей,

N – общее количество путей.

2) Если $c > k$, то i-й путь совпадает с первым на $k\%$ или более.

6. После нахождения всех путей, совпадающих с первым на $k\%$, из всего набора совпадающих путей можно оставить, например, самый короткий.
7. Если еще остались необработанные пути, повторить шаги 1-4 для них. Иначе – останов.
8. После нахождения всех путей, совпадающих с первым на $k\%$, из всего набора совпадающих путей можно оставить, например, самый короткий.
9. Если еще остались необработанные пути, повторить шаги 1-4 для них. Иначе – останов.

Алгоритм удаления схожих путей в своем изначальном варианте предполагал поиск пометки, которая содержит как можно больше путей, подлежащих удалению. Однако, как показал вычислительный эксперимент, даже в маленьком лабиринте в большинстве случаев алгоритм не мог найти такую пометку, потому что в ней также содержались один или несколько путей, которые удалять не нужно. По этой причине появляются так называемые пути-жертвы – это пути, которые закрываются вместе с теми путями, которые необходимо удалить.

4. Результаты и обсуждение

В ходе вычислительного эксперимента были поставлены следующие цели:

1. Выяснить, от каких входных параметров зависит количество получаемых проходов в лабиринте.
2. Найти оптимальное количество муравьев для муравьиного алгоритма при разных входных данных.
3. Оценить эффективность работы алгоритмов удаления путей.

Входными данными являются:

1. Размерность лабиринта.
2. Число вырезов, сделанных в алгоритме 3.
3. Интервал длин путей для алгоритма удаления по длине.

В Таблице 1 показано, что и количество путей в лабиринте, и оптимальное число муравьев зависит от вырезов, сделанных на 4 шаге модифицированного алгоритма Уилсона, и не зависит от размерности лабиринтов. Для нахождения закономерностей были взяты 3 лабиринта разных размерностей и с разным количеством дополнительных вырезов. Число вырезов равно 4, 6 и 8, число муравьев задано заведомо большое (более 2000). Если посмотреть на все строки с количеством вырезов равным 4, то можно увидеть, что при всех трех размерностях, количество полученных путей лежит в интервале $[3, 8]$. Максимальный номер последнего уникального пути можно увидеть при размерности 19×19 – он равен 49, кроме того, все номера также попадают в интервал $[3, 49]$ для всех размерностей.

Аналогично, если рассмотреть число дополнительных вырезов 4 и 6, можно выделить некоторые интервалы количества путей и номера последнего уникального пути, также не зависящие от размерности лабиринта: в случае 6 вырезов это число 154, а в случае 8 вырезов – 635.

Отсюда можно сделать вывод, что число полученных путей и оптимальное количество муравьев не зависят от размерности лабиринта, но зависят от количества дополнительных вырезов.

Также, исходя из полученных результатов, можно сделать вывод о том, что для 4 вырезов необходимо взять 49 муравьев, для 6 вырезов – 154 муравья и 635 муравьев для 8 вырезов. Однако рекомендуется брать некоторый «запас», к примеру, на 50 муравьев больше. Таким образом, получаем 99, 204 и 685 муравьев для 4, 6 и 8 вырезов соответственно. В Таблице 1 число муравьев округлено до 100, 200 и 700. Стоит отметить, что

при увеличении числа вырезов с 6 до 8, возможное количество муравьев возрастает более чем в три раза, что увеличивает время поиска всех путей.

Таблица 1 - Нахождение оптимального количества муравьев

Размерность лабиринта	Количество вырезов	Количество полученных путей	Номер последнего неповторяющегося пути	Возможное кол-во муравьев
19 × 19	4	4	12	100
		4	7	
		8	20	
		8	49	
		8	25	
19 × 19	6	14	40	200
		14	54	
		14	20	
		14	38	
		10	32	
19 × 19	8	48	421	700
		28	87	
		40	419	
		32	158	
		64	488	
55 × 55	4	8	9	100
		8	20	
		8	23	
		8	22	
		4	13	
55 × 55	6	12	41	200
		32	98	
		14	65	
		28	72	
		12	54	
55 × 55	8	28	259	700
		40	168	
		55	432	
		48	240	
		20	118	
179 × 179	4	4	7	100
		8	39	
		7	23	
		8	14	
		4	16	
179 × 179	6	12	45	200
		12	51	
		12	25	
		14	60	
		17	53	

		22	125	
		24	63	
		12	33	
		20	89	
179×179	8	30	250	700
		56	179	
		21	354	
		78	181	
		60	550	

В Таблице 2 приведены результаты работы алгоритма удаления по длине пути. Было построено 15 лабиринтов для трех размерностей, после чего для каждого были введены различные интервалы длин. При оценке эффективности работы алгоритма было выяснено, что почти в половине случаев удалось удалить 50% и более путей. Например, в лабиринте размерности 19×19 при интервале $[38, 45]$ алгоритм удалил абсолютно все пути, длины которых оказались вне введенного интервала, но также в некоторых случаях алгоритм не удалил ни одного пути, например, при размерности 55×55 и интервале $[240, 290]$. Таким образом, при грамотно подобранном интервале длин эффективность алгоритма составляет более 50%.

Таблица 2 - Результаты работы алгоритма удаления по длине пути

Размерность лабиринта	Кол-во путей	Интервал длин путей	Введенный интервал	Количество путей, подлежащих удалению	Количество удаленных путей
19x19	9	[38,74]	[38,55]	5	3
19x19			[38,45]	6	6
19x19			[50,65]	6	3
19x19			[55,74]	4	3
19x19			[65,74]	7	6
55x55	21	[162,350]	[162,190]	14	9
55x55			[162,250]	11	6
55x55			[200,300]	10	0
55x55			[240,290]	14	0
179x179			[300,350]	18	7
179x179	28	[590,1698]	[590,1000]	22	8
179x179			[590,1400]	6	0
179x179			[950,1300]	22	8
179x179			[1200,1698]	10	0

179x179			[1400,1698]	22	7

Таблица 3 показывает результаты работы алгоритма удаления схожих путей при $k = 80$. Для трех размерностей было построено по 5 различных лабиринтов, к которым был применен алгоритм удаления схожих путей. В Таблице 3 наибольший интерес представляют три последних столбца. Например, в трех первых строках можно увидеть, что из всех путей, которые необходимо удалить, алгоритм не удалил ни одного. В четвертой же строке другая ситуация: вместо положенных 6 путей алгоритм смог удалить только 4, однако даже для этого пришлось удалить 3 пути-жертвы. И лишь в одном из случаев размерности 55×55 с количеством путей, разным 10, алгоритм удалил все 2 необходимых пути, но вдобавок убрал один путь-жертву.

Таблица 3 - Результаты работы алгоритма удаления схожих путей

Размерность лабиринта	Кол-во путей	Количество путей, подлежащих удалению	Количество удаленных путей	Кол-во путей-«жертв»
19×19	24	4	0	0
	22	4	0	0
	14	1	0	0
	14	6	4	3
	14	2	0	0
55×55	14	3	2	1
	24	8	4	4
	10	2	2	1
	21	10	6	3
	12	3	1	1
179×179	7	2	1	1
	14	2	0	0

	24	12	8	4
	21	7	4	5
	8	6	4	

5. Заключение

В статье предложены модификации алгоритма Уилсона, муравьиного алгоритма, а также алгоритмы удаления по длине путей и удаления схожих путей. Они расширяют возможности генерации лабиринтов, а именно, позволяют регулировать количество всех путей при генерации, находить все существующие в лабиринте проходы, а также редактировать полученные лабиринты двумя способами. Все это дает более гибкий процесс генерации.

Проведен вычислительный эксперимент, в ходе которого удалось установить зависимость количества путей от параметров генерации лабиринта, оптимальное количество муравьев для нахождения всех путей в лабиринте. Два алгоритма редактирования путей в лабиринте были протестированы и исследованы на эффективность, были выявлены их недостатки.

В дальнейшем планируется оптимизировать процессы генерации и нахождения путей, а также разработать новые алгоритмы редактирования лабиринта.

ЛИТЕРАТУРА

1. Классические алгоритмы генерации лабиринтов. Часть 1: вступление. – URL: https://habr.com/post/320140/ .
2. Классические алгоритмы генерации лабиринтов. Часть 2: погружение в случайность. – URL: https://habr.com/post/321210/ .
3. Walter P. Maze Algorithms / P. Walter // Astrolog – URL: http://www.astrolog.org/labyrnth/algrithm.htm .
4. Штовба С. Д. Муравьиные алгоритмы / С. Д. Штовба // Exponenta Pro. – 2003. – №4. – С. 70-75.

E.S. Simonyan, O.A. Medvedeva, S.N. Medvedev

CREATION OF MAZE WITH MULTIPLE SOLUTIONS, SEARCH FOR ALL SOLUTIONS AND EDITING THEM

Voronezh State University,
Voronezh, Russia

This article considers the problem of creating a maze with multiple solutions, searching for all solutions and a possibility of editing them. First, we view two algorithms, which generate the most various mazes, and find their positive and negative sides. Then we choose the one most efficient and suitable for the problem. All maze creation algorithms, including two considered, are unable to reveal a definite number of solutions. Here we offer a modification of Wilson's algorithm. Still we tend to think that all maze solving algorithms cannot find all possible solutions. Therefore, we develop a modification of Ant Colony algorithm. To edit mazes the two algorithms were developed: one by solutions lengths and another by similarity of solutions. The research was conducted in order to find dependencies between number of solutions and initial parameters. A computational experiment was made to find optimal number of ants within different initial parameters. Also, there were evaluated efficiencies of two maze editing algorithms with different parameters.

Keywords: maze generation, Ant Colony algorithm, Wilson's algorithm, computational experiment.

REFERENCES

1. Classic maze generation algorithms. Part 1: introduction. – URL: https://habr.com/post/320140/ (дата обращения 20.11.2017).
2. Classic maze generation algorithms. Part 2: immersion in random. – URL: https://habr.com/post/321210/ .
3. Walter P. Maze Algorithms / P. Walter // Astrolog – URL: http://www.astrolog.org/labyrnth/algrithm.htm .

Shtovba S. D. Ant Colony algorithms / S. D. Shtovba // Exponenta Pro. – 2003. – №4. – P. 70-75