Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Магистерская программа «Логические и комбинаторные методы анализа данных»

Магистерская диссертация

Метод латентных ребер для слияния перекрывающихся триангуляций Делоне

Выполнила:
студентка 617 группы
Готман Мария Леонидовна
подпись студента

Научный руководитель:
д.т.н., профессор
Местецкий Леонид Моисеевич
подпись научного руководителя

Москва, 2017

Содержание

Аннотация

Рассматривается задача слияния двух триангуляций Делоне, причем множества, на которых заданы исходные триангуляции, допускают полное перемешивание. Вычислительная сложность задачи построения триангуляции Делоне в общем случае составляет $O(n \log n)$. Если ослабить условие задачи и рассматривать линейно разделенные триангуляции, тогда слияние можно произвести за $O(n)$. Существуют алгоритмы [1–3], позволяющие решить поставленную задачу за время $O(n)$, однако они используются сложные структуры данных, как диаграммы Вороного и минимальные остовные деревья.

В данной работе предложен алгоритм, который позволяет отказаться от использования сложных структур данных, причем его вычислительная эффективность экспериментально подтверждена.

1 Введение

Впервые задача построения триангуляции Делоне была рассмотрена в 1934 году российским математиком Борисом Делоне. Задача построения триангуляции Делоне (ТД) n точек-сайтов имеет нижнюю оценку вычислительной сложности $O(n \log n)$ [4, стр.192-195], как задача, обладающая такой же вычислительной сложностью, что и задача сортировки. Эту оценку можно улучшить, если имеется некоторая дополнительная информация о структуре и связях сайтов.

В 1980 году Ли и Шехтером была рассмотрена задача слияния линейно разделенных триангуляций Делоне [5] методом «разделяй и властвуй». Исходное множество точек рекурсивно делится на два линейно разделимых подмножества. Рекурсия прекращается, когда в подмножестве останется по две или три точки на которых строится тривиальная триангуляция. Каждый шаг слияния разделенных триангуляций имеет вычислительную сложность $O(n)$. Общая сложность такого алгоритма составляет $O(n \log n)$.

Мы рассматриваем задачу слияния двух триангуляций Делоне, когда исходное множество сайтов состоит из двух непересекающихся подмножеств $\mathbf{S} = \mathbf{S_1} \cup \mathbf{S_2}$, $\mathbf{S_1} \cap \mathbf{S_2} = \emptyset$, при том, что выпуклые оболочки этих множеств перекрываются $Conv(\mathbf{S_1}) \cap Conv(\mathbf{S_2}) \neq \emptyset$. В качестве входных данных также имеем триангуляции Делоне $Del(\mathbf{S_1})$ и $Del(\mathbf{S_2})$ исходных множеств $\mathbf{S_1}$ и $\mathbf{S_2}$ соответственно. Встает вопрос, если существует алгоритм слияния линейно разделенных триангуляций за время $O(n)$, можно ли построить алгоритм, который будет иметь такую же асимптотику, но позволит производить слияние линейно неразделенных триангуляций.

Такую задачу можно решить, используя диаграммы Вороного (ДВ). Диаграмма Вороного n сайтов является двойственным графом к триангуляции Делоне. Эти структуры данных могут быть получены одна из другой за время $O(n)$. Алгоритм Киркпатрика [1] строит ДВ $Vor(\mathbf{S_1} \cup \mathbf{S_2})$ перекрывающихся множеств сайтов $\mathbf{S_1}$ и $\mathbf{S_2}$ на основе слияния ДВ $Vor(\mathbf{S_1})$ и $Vor(\mathbf{S_2})$. Для решения нашей задачи нужно преобразовать исходные ТД $Del(\mathbf{S_1})$, $Del(\mathbf{S_2})$ в $Vor(\mathbf{S_1})$, $Vor(\mathbf{S_2})$ соответственно, построить $Vor(\mathbf{S_1} \cup \mathbf{S_2})$ с помощью алгоритма [1], а затем преобразовать $Vor(\mathbf{S_1} \cup \mathbf{S_2})$ в $Del(\mathbf{S_1} \cup \mathbf{S_2})$. Другой подход к решению данной задачи используется в алгоритме Шазеля [2]. Предлагается построить пересечение двух выпуклых многогранников в 3D-пространстве. Для его использования применительно к нашей задаче требуется построить из исходных ТД $Del(\mathbf{S_1})$, $Del(\mathbf{S_2})$ сначала ДВ $Vor(\mathbf{S_1})$, $Vor(\mathbf{S_2})$, затем выпуклые многогранники. Затем нужно построить пересечение этих многогранников с помощью алгоритма [2]. После этого произвести обратно преобразование в триангу-

ляцию Делоне. Таким образом, используя алгоритмы [1,2], мы получаем решение за 3 и 5 шагов соответственно. И хотя вычислительная сложность этих решений остается $O(n)$, их практическая реализация весьма проблематична.

Ранее уже были попытки [3, 6] решения задачи в такой постановке. Однако эти решения использовали построение минимальных остовных деревьев, что довольно трудоемко при реализации. Целью данной работы является разработка вычислительно эффективного алгоритма слияния перекрывающихся триангуляций Делоне, не используя минимальные остовные деревья.

Возможным практическим применением данного алгоритма служит вычисления расстояний в метрическом пространстве функций двух переменных, заданных на конечных нерегулярных множествах точек. Эта задача возникает, в частности, при анализе 3D поверхностей человеческих лиц, полученных в результате пространственного сканирования [7]. Предложенная модель определяет расстояние между такими функциями на основе построения общей ТД. При этом расстояние между моделями лиц определяется как минимальное по всем движениям множеств $\mathbf{S_1}$ и $\mathbf{S_2}$ на плоскости. Таким образом, осуществляется подгонка пары поверхностей друг к другу. И на каждой итерации подгонки выполняется слияние триангуляций.

2 Терминология и постановка задачи

Пусть на евклидовой плоскости задано $\mathbf{S}$ — множество из $n \geq 3$ *точек-сайтов*, не все из которых лежат на одной прямой.

Триангуляцией конечного множества точек $\mathbf{S}$ называется планарный граф с вершинами из $\mathbf{S}$, все внутренние области которого являются треугольниками. Триангуляция называется *выпуклой*, если минимальный многоугольник, содержащий все ее треугольники, будет выпуклым. Далее под термином *грань* будем понимать только конечную треугольную грань триангуляции. Ребро и грань называются *инцидентными*, если они имеют две общие вершины. Ребра, инцидентные одной грани, называются *смежными*. Сайт называется *инцидентным* ребру, если он является одним из концов рассматриваемого ребра.

Окружность называется *пустой*, если она не содержит внутри себя сайтов. Прямая линия, по одну сторону от которой нет сайтов, называется *несобственной* пустой окружностью. Окружность, проходящая через сайт, называется *инцидентной* этому сайту. *Ребром Делоне* называется ребро, инцидентные сайты которого имеют общую пустую инцидентную окружность. *Гранью Делоне* называют грань, вершины которой имеют общую пустую инцидентную окружность.

Следующие определения эквивалентны:

Определение. *Триангуляцией Делоне (ТД)* $Del(\mathbf{S})$ множества точек-сайтов $\mathbf{S}$ называется выпуклая триангуляция, у которой описанная окружность каждой треугольной грани является пустой, т.е. все грани которой являются гранями Делоне.

Определение. *Триангуляцией Делоне* множества точек-сайтов $\mathbf{S}$ называется выпуклая триангуляция, у которой для каждого ребра существует пустая инцидентная сайтам-вершинам окружность, т.е. все ребра которой являются ребрами Делоне.

Далее в настоящей работе под термином триангуляция будет пониматься триангуляция Делоне.

Задача слияния двух перекрывающихся триангуляций Делоне ставится следующим образом. Даны два конечных линейно неразделимых множества сайтов $\mathbf{B}$ и $\mathbf{W}$ (считаем, что сайты раскрашены в два цвета: множество черных сайтов $\mathbf{B}$ и множество белых сайтов $\mathbf{W}$), а также их триангуляции Делоне $Del(\mathbf{B})$ и $Del(\mathbf{W})$. Нужно построить триангуляцию Делоне на объединенном множестве сайтов $Del(\mathbf{B} \cup \mathbf{W})$.

Пример исходных данных и искомой триангуляции приведен на рис. 1.

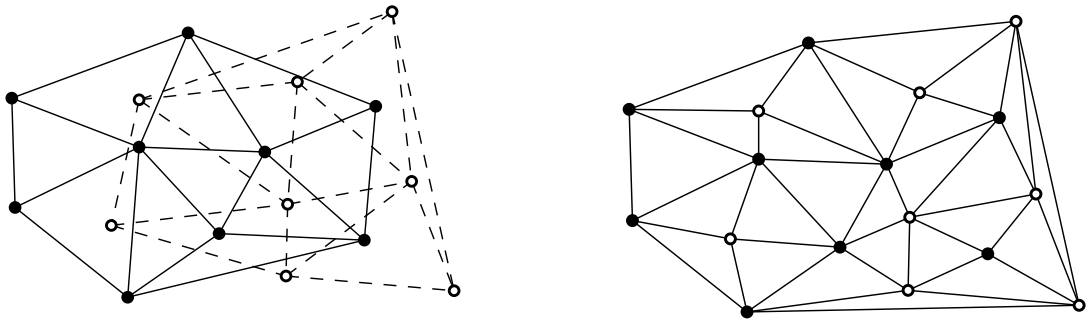


Рис. 1: Исходные триангуляции и объединенная триангуляция

Существует наивное решение данной задачи. Не используя информацию об исходных триангуляциях, построить триангуляцию на объединенном множестве $\mathbf{B} \cup \mathbf{W}$. Вычислительная сложность такого подхода составляет $O(n \log n)$. В данной работе мы хотим, используя дополнительно исходные триангуляции, уменьшить вычислительную сложность задачи и добиться упрощения алгоритма по сравнению с существующими решениями.

3 Структура данных

В нашей задаче на вход алгоритму подаются множества точек $\mathbf{B}$ и $\mathbf{W}$, а также их исходные триангуляции Делоне. Множество $\mathbf{B}$ содержит n_1 точек с координата-

ми $(x_{11}, y_{11}),\ (x_{12}, y_{12}),\ \ldots,\ (x_{1n_1}, y_{1n_1})$, множество $\mathbf{W}$ — n_2 точек с координатами $(x_{21}, y_{21}),\ (x_{22}, y_{22}),\ \ldots,\ (x_{2n_2}, y_{2n_2})$:

$$\begin{aligned} \mathbf{B} &= \{(x_{11}, y_{11}),\ (x_{12}, y_{12}),\ \ldots,\ (x_{1n_1}, y_{1n_1})\} \\ \mathbf{W} &= \{(x_{21}, y_{21}),\ (x_{22}, y_{22}),\ \ldots,\ (x_{2n_2}, y_{2n_2})\} \end{aligned} \tag{1}$$

Объединенное множество точек $\mathbf{B} \cup \mathbf{W}$, содержащее $n = n_1 + n_2$ точек, будем обозначать:

$$\mathbf{P} = \mathbf{B} \cup \mathbf{W} \tag{2}$$

Структура данных, представляющая собой триангуляции множеств точек $\mathbf{B}$ и $\mathbf{W}$, может содержать информацию о ребрах и/или гранях соответствующей исходной триангуляции. Выбор структуры данных не влияет на алгоритмическую сложность задачи, потому что переход от одной структуры данных к другой осуществляется за линейное время. Однако это сильно влияет на удобство использования алгоритма.

Пучком сайта $p \in \mathbf{P}$ называют множество ребер триангуляции, инцидентных сайту p. Пучок сайта будем представлять в виде двунаправленного циклического списка сайтов $p_1,\ \ldots,\ p_q$, соединенных ребром с p в триангуляции, так чтобы они шли в направлении обхода против часовой стрелки относительно p.

В предложенном алгоритме будут часто производиться коррекции пучков сайтов, поэтому операция добавления и удаления ребер из пучка должна выполняться максимально эффективно. Поэтому для нашего решения будем использовать структуру данных *«Узлы с соседями»*. Эта структура данных для каждого сайта p хранит список сайтов, входящих в его пучок, то есть содержит в себе ребра триангуляции.

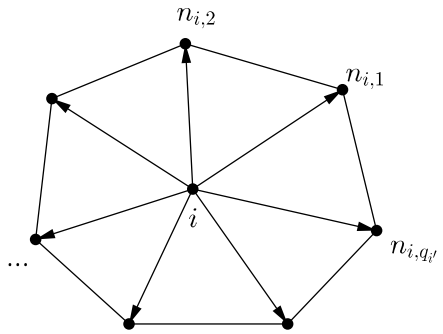


Рис. 2: Структура данных *«Узлы с соседями»*

Для хранения описанной структуры данных для каждого сайта создается циклический список сайтов, принадлежащих его пучку в порядке обхода против часовой стрелки:

$$\mathbf{N} = \{neighbors_1,\ neighbors_2,\ \ldots,\ neighbors_n\}, \tag{3}$$

где $neighbors_i = \{n_{i,1},\ n_{i,2},\ \ldots,\ n_{i,q_{i'}}\}$, i' — количество точек, попавших в пучок i-ого сайта (рис. 2).

Пусть сайт с номером j предшествует сайту с номером k в пучке сайта с номером i, тогда верно:

$$\begin{aligned} j &= neighbors_i.\mathbf{prev}(k) \\ k &= neighbors_i.\mathbf{next}(j) \end{aligned} \tag{4}$$

4 Описание метода латентных ребер

В объединенной триангуляции Делоне $Del(\mathbf{B} \cup \mathbf{W})$ часть ребер переходит из исходных ТД в результирующую ТД. В процессе построения результирующей триангуляции строятся новые разноцветные ребра, при этом часть ребер исходных триангуляций перестает удовлетворять условию Делоне на объединенном множестве точек, и должна быть разрушена.

Латентным ребром называется разноцветная пара сайтов, для которой выполнено условие Делоне по всему множеству сайтов $\mathbf{B} \cup \mathbf{W}$. *Дефектным ребром* называется одноцветная пара сайтов, образующая ребро Делоне в $Del(\mathbf{B})$ или $Del(\mathbf{W})$, для которой не выполнено условие Делоне на множестве $\mathbf{B} \cup \mathbf{W}$. *Открытым ребром* называется латентное ребро, имеющее не более одной инцидентной грани, не лежащее в выпуклой оболочке ТД, либо не имеющее инцидентных граней и лежащее в выпуклой оболочке ТД. Сайт называется *открытым*, если для него существует еще не построенное открытое ребро. *Швом* называется максимальная связная последовательность латентных ребер. *Стартером* называется любое открытое ребро, еще не включенное в объединенную триангуляцию, от которого начинается построение шва.

Суть метода заключается в следующем: нужно построить стартер, от которого начать построение шва. Когда построение шва будет закончено, приступить к поиску очередного стартера. Если стартер найден, то продолжить от него построение шва, иначе считаем процесс построения объединенной триангуляции законченным.

Далее подробно будет описан каждый шаг алгоритма.

4.1 Условие существования стартера

Рассмотрим лемму 1, описывающую условие существования латентного ребра для сайта.

Лемма 1. *Открытый сайт имеет инцидентное латентное ребро $\Leftrightarrow$ существует инцидентная ему окружность, внутри которой нет сайтов одного с ним цвета, но есть сайты противоположного цвета на ней или внутри нее.*

Доказательство. Достаточность. Пусть для некоторого сайта A существует инцидентная ему окружность с центром C, внутри которой есть только сайты противоположного с A цвета $D_1,\ D_2,\ \ldots,\ D_k,\ k \geq 1$ внутри или на окружности (рис. 3). Рассмотрим множество окружностей, инцидентных парам сайтов A и $D_i,\ i = 1,\ \ldots,\ k$, имеющих в точке A общую касательную с окружностью с центром в точке C. Таким образом строим заданную пустую окружность минимального радиуса, инцидентную разноцветным сайтам A и D_{min}. Значит, эта пара сайтов образует латентное ребро.

Необходимость. Есть пара сайтов, образующая латентное ребро с инцидентной пустой окружностью. Эта окружность и будет удовлетворять условию леммы. ■

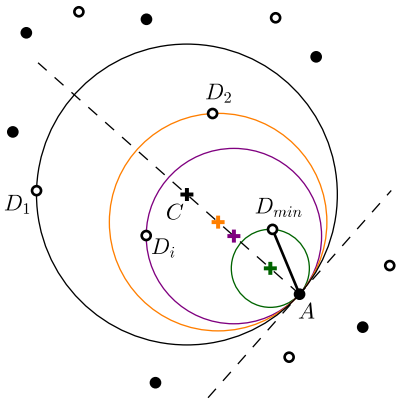


Рис. 3: Пояснение к лемме 1

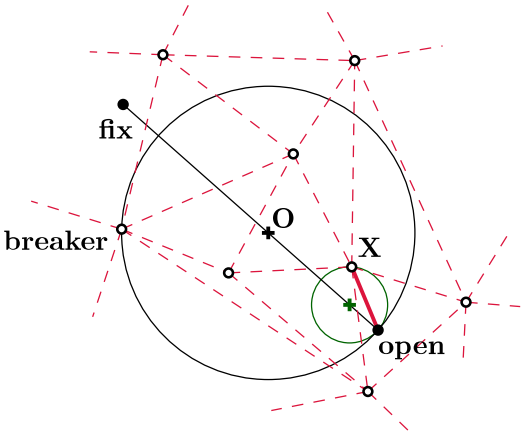


Рис. 4: Пояснение к алгоритму 1

На основании леммы составим алгоритм 1 построения стартера, которому на вход подаются три точки:

- первая точка является открытой, лежит на окружности **O** из условия леммы 1,
- вторая точка лежит на прямой, соединяющей открытую точку и центр окружности **O**, назовем ее *фиксированной*,
- третья — противоположного с первой цвета, также лежит на окружности **O**, назовем ее *нарушителем*.

Не ограничивая общности будем считать открытую точку черного цвета, тогда внутри окружности **O** находятся только точки белого цвета. Обходом в ширину, начиная с нарушителя, пройдем по всем точкам графа исходной триангуляции множества **W**, которые попали внутрь **O**. Среди таких точек находим сайт **X**, образующий с открытым сайтом окружность минимального радиуса с центром на прямой, проходящей через открытую и фиксированную точки (см. лемму 1). Тогда ребро, составленное из открытого сайта и сайта **X**, будет являться стартером.

Алгоритм 1 Построение стартера

```
1: function CALCULATESTARTER(open, fix, breaker)
2:     d — прямая, проходящая через open и fix
3:     starter_0 = open
4:     R_min = ∞
5:     checkPoints := breaker
6:     visited := ∅
7:     while checkPoints ≠ ∅ do
8:         x := checkPoints.get()
9:         if x ∉ visited then
10:            r — радиус окружности с центром на d, инцидентной сайтам open и x
11:            if r < R_min then
12:                R_min := r
13:                starter_1 := x
14:            end if
15:            for p ∈ neighbours_x do
16:                if p лежит внутри окружности O then
17:                    checkPoints.add(p)
18:                end if
19:            end for
20:        end if
21:    end while
22:    return [starter_0, starter_1]
23: end function
```

4.2 Поиск первого стартера

Поиск первого стартера будем осуществлять на основе леммы 1. Будем говорить, что сайт A *лежит левее* сайта B, если A предшествует B при лексикографическом упорядочивании сайтов по возрастанию координат $(x,\ y)$.

Утверждение 1. *Пусть сайты B_{min} и W_{min} — самые левые точки множеств* **B** *и* **W** *соответственно. Тогда существует стартер, инцидентный одному из этих сайтов.*

Доказательство. Не ограничивая общности, будем считать, что B_{min} лежит левее, чем W_{min} (рис. 5). Рассмотрим окружность O, инцидентную сайтам B_{min} и W_{min} с центром на луче, параллельном оси Ox, выходящем из точки W_{min} влево. Окружность O не содержит ни одного сайта из **W**, потому что лежит левее самого левого

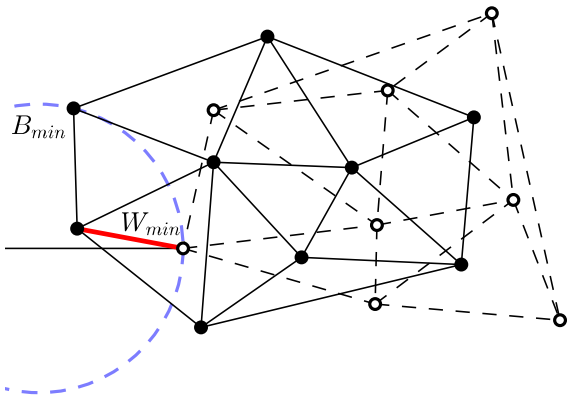


Рис. 5: Пояснение к утверждению 1

сайта множества W_{min}, и содержит сайт B_{min} на границе окружности. Тогда выполнены все уловия леммы 1, и существует стартер, инцидентный W_{min}.

Если точки B_{min} и W_{min} лежат на одной вертикальной прямой, то окружность окажется несобственной, тогда среди всех точек, лежащих на этой вертикальной прямой выберем ближайшую к W_{min}, эта пара и будет образовывать искомый стартер. ■

На основе утверждения 1 составим формальный алгоритм 2, с помощью которого можно производить поиск первого стартера.

Алгоритм 2 Поиск первого стартера

```
1: function FIRSTSTARTER()
2:     B_min — самая левая точка множества B
3:     W_min — самая левая точка множества W
4:     starter_0 := W_min          ▷ Не ограничивая общности, B_min лежит левее W_min
5:     W'_min — точка, сдвинутая по оси Ox влево относительно W_min
6:     starter_1 := CALCULATESTARTER(W_min, W'_min, B_min)
7:     return [starter_0, starter_1]
8: end function
```

4.3 Удаление дефектных ребер

При построении латентных ребер часть ребер исходных триангуляций перестает удовлетворять условию Делоне для ребер, и должна быть разрушена для корректного построения результирующей триангуляции. Рассмотрим лемму, помогающую проверить условие Делоне для заданного ребра:

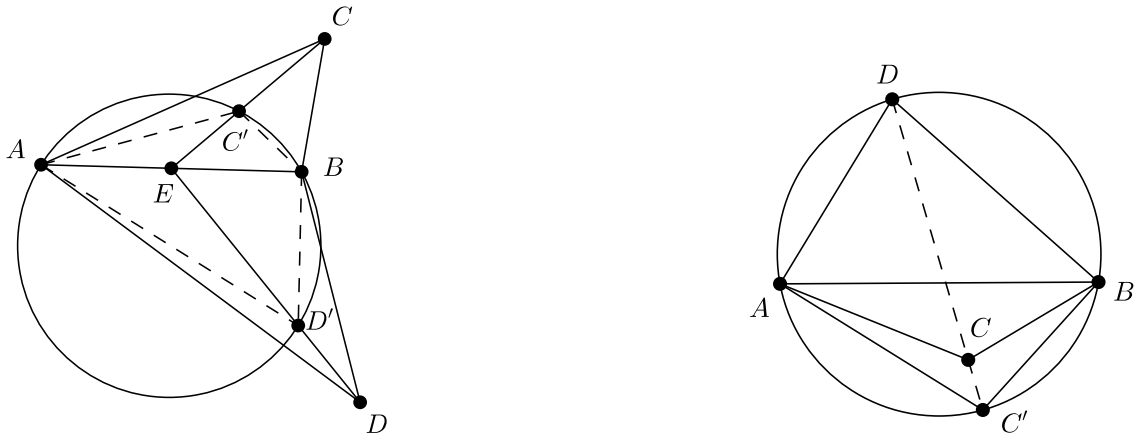


Рис. 6: Пояснение к лемме 2

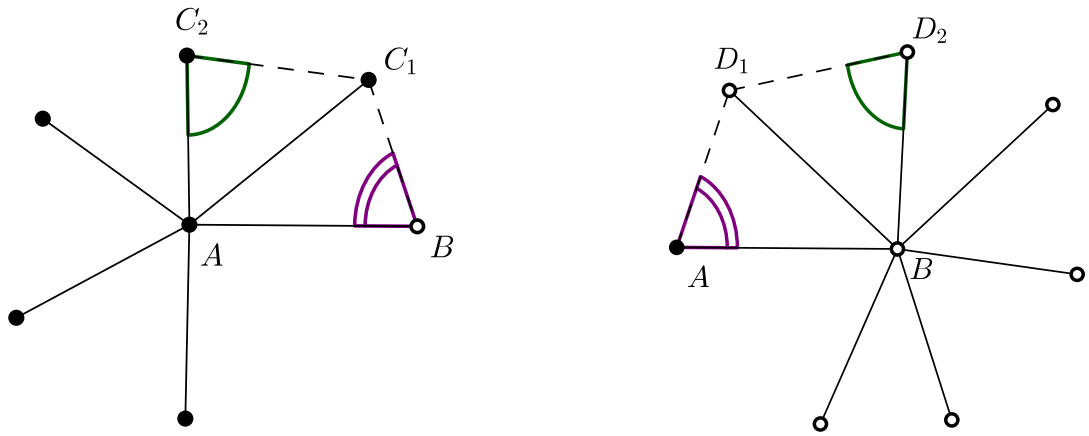
Лемма 2. *(Угловой критерий) Пусть* $\mathbf{S}$ *— множество сайтов, для пары* $A,\ B \in \mathbf{S}$ *выполнено условие Делоне, тогда для любых других двух сайтов* $C,\ D \in \mathbf{S}$*, лежащих по разные стороны от* AB*, справедливо:*

(a) $\angle ACB + \angle ADB \leq 180°$*;* *(b)* $\angle ABD \geq \angle ACD$*.*

Доказательство. Докажем утверждение (a). Существует пустая окружность, инцидентная сайтам A и B (рис. 6a). Пусть E — середина AB. Отрезки EC и ED пересекают окружность в точках C' и D' соответственно. В четырехугольнике $AC'BD'$, вписанном в окружность, имеем $\angle AC'B + \angle AD'B = 180°$. Если окружность пустая, то $\angle ACB \leq \angle AC'B$ и $\angle ADB \leq \angle AD'B$. Тогда получаем: $\angle ACB + \angle ADB \leq \angle AC'B + \angle AD'B = 180°$.

Докажем утверждение (b). Предположим противное, пусть $\angle ABD < \angle ACD$. Построим описанную окружность для треугольника $\triangle ABD$ (рис. 6b). Тогда C лежит внутри нее. Пусть C' — точка пересечения прямой CD с окружностью. В четырехугольнике $AC'BD$, вписанном в окружность, имеем $\angle AC'B + \angle ADB = 180°$. Поскольку $\angle AC'B < \angle ACB$, получаем, что $\angle ACB + \angle ADB > 180°$. А тогда для AB согласно первому утверждению леммы не выполняется условие Делоне. Значит, предположение $\angle ABD < \angle ACD$ неверно, и утверждение леммы выполняется. ■

Пусть найдена пара сайтов A и B, образующая латентное ребро, ещё не включенное в объединенную триангуляцию. Присоединение ребра AB к пучкам сайтов-вершин A и B может привести к нарушению условия Делоне для некоторых ребер из этих пучков, то есть часть ребер окажутся дефектными. Все дефектные ребра должны быть удалены сразу после присоединения нового латентного ребра.



(a) (b)

Рис. 7: Удаление дефектных ребер

Пусть AC_1 и AC_2 одноцветные ребра пучка A, что сайты B, C_1 и C_2 идут в порядке обхода против часовой стрелки в пучке сайта A (рис. 7a). Для ребра AC_1 нарушено условие Делоне, если $\angle AC_2C_1 + \angle ABC_1 > 180°$ (лемма 2). В этом случае ребро AC_1 удаляется из объединенной триангуляции, а ребро AC_2 подвергается аналогичной проверке. Если же для ребра AC_1 условие Делоне выполнено, то ребро AC_1 сохраняется в объединенной триангуляции, проверка условия Делоне для остальных ребер не производится.

Аналогичным образом выполняется коррекция пучка сайта B в направлении обхода по часовой стрелке (рис. 7b).

После этого производим коррекцию ребра AB в противоположном направлении, то есть точки B, C_1 и C_2 идут в порядке обхода по часовой стрелке в пучке сайта A, пучок сайта B корректируем против часовой стрелки. Таким образом, пучки сайтов A и B будут скорректированы как по часовой стрелке, так и против нее. На этом вставка нового латентного ребра AB заканчивается.

Все дефектные ребра занесем в отдельный список *dumping* вместе с точкой, которая нарушает условие леммы 2. Ребро AC_1 удовлетворяло условию Делоне в исходной триангуляции, а при добавлении ребра AB перестало удовлетворять ему. Значит, точка B является *нарушителем* условия Делоне для ребра AC_1, и будет сохранена в *dumping* при удалении этого ребра. Аналогично при удалении ребра BD_1 сохраняем точку A. Эта информация понадобится при дальнейшем поиске стартеров. Об этом подробно будет рассказано ниже.

Приведем формальное описание процедуры коррекции пучков сайтов (алгоритм 3). Параметрами являются:

- сайты A и B, образующие латентное ребро,
- *reverse* является внутренним параметром. После того, как для ребра AB будет произведена коррекция пучков в направлении по часовой стрелке, ребро AB разворачивается в ребро BA, и продолжается коррекция пучков в противоположном направлении.

Алгоритм 3 Удаление дефектных ребер

```
1: procedure DELETEDEFECTEDGES(A, B, reverse = False)
2:     C_1 := neighbors_A.prev(B)                    ▷ Обрабатываем пучок сайта A
3:     C_2 := neighbors_A.prev(C_1)
4:     while ∠ABC_1 + ∠AC_2C_1 > 180° do
5:         dumping.add({AC_1, B})
6:         Удалить ребро AC_1 из триангуляции
7:         _1 := _2
8:         _2 := neighbors_A.prev(C_2)
9:     end while
10:    D_1 := neighbors_B.next(A)                    ▷ Обрабатываем пучок сайта B
11:    D_2 := neighbors_B.next(D_1)
12:    while ∠BAD_1 + ∠BD_2D_1 > 180° do
13:        deleteEdges.add({BD_1, A})
14:        Удалить ребро BD_1
15:        D_1 := D_2
16:        D_2 := neighbors_B.next(D_2)
17:    end while
18:    if reverse == False then      ▷ Развернуть ребро AB, продолжить проверку
19:        DELETEDEFECTEDGES(B, A, True)
20:    end if
21: end procedure
```

4.4 Построение латентных ребер

Пучки, для которых выполнена описанная коррекция, будем называть *правильными*. Рассмотрим алгоритм образования новых латентных ребер объединенной ТД:

Лемма 3. *Пусть AB открытое ребро, пучки сайтов A и B являются правильными. Пусть C следует за точкой B в пучке сайта A, точка D предшествует точке A в*

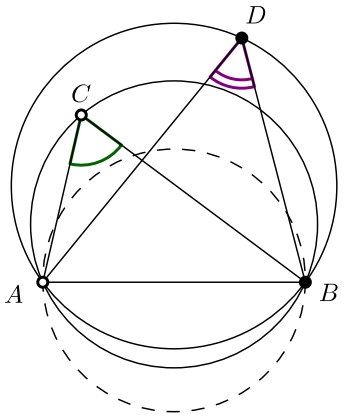


Рис. 8: Пояснение к лемме 3

пучке сайта B, тогда если $\angle ACB > \angle ADB$, то BC является новым латентным ребром, иначе AD является новым латентным ребром Делоне.

Доказательство. Поскольку AB ребро Делоне, то существует пустая окружность, инцидентная сайтам A и B (рис. 8). Значит, сайты C и D лежат вне этой окружности. Рассмотрим описанные окружности треугольников $\triangle ACB$ и $\triangle ADB$. Очевидно, что дуги этих окружностей, лежащие ниже AB, целиком попадают внутрь пустой окружности ребра Делоне AB, следовательно, снизу от AB внутри этих окружностей сайты лежать не могут. А выше ребра AB одна из этих окружностей также является пустой, причем та, у которой угол треугольника, лежащий против стороны AB, больше (если углы $\angle ACB$ и $\angle ADB$ равны, то можно выбрать любое ребро). Это вытекает из правильности пучков сайтов A и B. ■

Утверждение 2. *Если AB является открытым ребром, то существует C в пучке сайта A или D в пучке сайта B, что AD или BC является латентным ребром.*

Доказательство. Не ограничивая общности будем считать, что ребро AB не имеет инцидентной грани сверху (в противном случае перейдем к рассмотрению ребра BA). По лемме 3 для открытого ребра существует следующее латентное ребро. ■

В процессе построения латентных ребер в любой момент времени открытых ребер не более двух (стартер и новое латентное ребро). Пока очередное латентное ребро является открытым, продолжаем построение последующих латентных ребер, и удаление дефектных ребер, соответствующих новому латентному ребру. Если очередное ребро перестало являться открытым, то оно совпало со вторым открытым ребром, в этом случае процесс построения латентных ребер нужно приостановить, либо оказалось ребром выпуклой оболочки, тогда нужно продолжать построение латентных

ребер от стартера (оставшегося открытого ребра) в противоположную сторону, пока очередное ребро не будет являться ребром выпуклой оболочки. На этом приостановим процесс построения латентных ребер, так как для его запуска требуется найти стартер. Процедуру, описывающую этот процесс, представим в алгоритме 4. На вход алгоритму подаются точки, образующие стартер в объединенной триангуляции.

4.5 Поиск последующих стартеров

Поиск последующих стартеров основывается на использовании информации о дефектных ребрах. Чтобы в дальнейшем оценить вычислительную сложность алгоритма, обратимся к теории использования минимальных остовных деревьев. Как увидим далее, на практике можно будет отказаться от них и использовать более слабое условие, что значительно облегчит понимание и реализацию предложенного алгоритма.

4.5.1 Минимальные остовные деревья

Минимальным остовным деревом (МОД) триангуляции Делоне называется ее связный подграф, имеющий наименьшую суммарную длину ребер. Пусть на плоскости задано n точек. *Евклидовым МОД (ЕМОД)* называется связный подграф, вершинами которого являются все n точек, суммарная длина всех ребер которого минимальна. Известно, что МОД ТД является ЕМОД для множества сайтов ТД [4, стр. 227, 277].

МОД исходных триангуляций вычисляются один раз, поэтому это действие можно отнести к этапу предобработки, предшествующему непосредственному слиянию исходных триангуляций. Теоретическая оценка трудоемкости задачи построения минимального остова составляет $O(n \log n)$. Поскольку известно, что МОД является подграфом триангуляции Делоне (рис. 9), и на ее основе МОД может быть построен за линейное время. Такой вычислительной сложностью обладает алгоритм Чериона-Тарьяна [4, стр. 226-230].

Поиск последующих стартеров основан на том, что при удалении дефектных ребер нарушается связность исходных триангуляций, поэтому среди разрушенных ребер обязательно окажется ребро МОД этой ТД. *Мостами* будем называть дефектные ребра исходных триангуляций, входящие в МОД.

Будем называть окружность, диаметром которой является ребро МОД, *окружностью влияния ребра МОД*. Леммы 4 и 5 описывают свойства МОД, которые понадобятся для оценки вычислительной сложности алгоритма.

Алгоритм 4 Построение шва от стартера

```
1: procedure constructLatent(A_0, B_0, reverse = False)
2:     [A, B] := [A_0, B_0]
3:     while True do
4:         deleteDefectEdges(A, B)
5:         C := neighbors_A.prev(B)
6:         D := neighbors_B.next(A)
7:         if ∠ACB ≥ ∠ADB  &  ∠ACB < π then          ▷ CB — новое ребро
8:             if [C, B] ≠ [A_0, B_0] then
9:                 Добавить ребро CB в триангуляцию
10:                A := C
11:            else
12:                reverse := True                  ▷ Новое ребро совпало с стартером
13:                break
14:            end if
15:        else if ∠ADB > ∠ACB  &  ∠ADB < π then     ▷ AD — новое ребро
16:            if [A, D] ≠ [A_0, B_0] then
17:                Добавить ребро AD в триангуляцию
18:                B := D
19:            else
20:                reverse := True                  ▷ Новое ребро совпало с стартером
21:                break
22:            end if
23:        else
24:            break
25:        end if
26:    end while
27:    if reverse == False then      ▷ Построение шва в противоположную сторону
28:        constructLatent(B_0, A_0, True)
29:    end if
30: end procedure
```

Лемма 4. *Окружность влияния ребра МОД является пустой окружностью триангуляции, то есть не содержит внутри себя сайтов.*

Доказательство. Предположим противное: пусть AB — ребро МОД и внутрь окружности с диаметром AB попадает точка C (рис. 10). В $\triangle ABC$ угол $\angle C$ тупой, поэтому

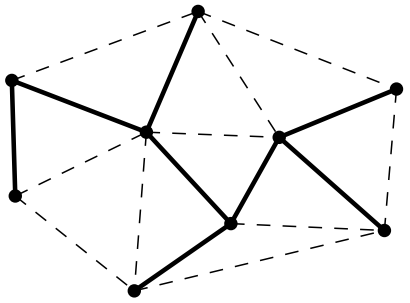


Рис. 9: Евклидово минимальное остовное дерево триангуляции Делоне

$|AC| < |AB|$ и $|BC| < |AB|$. Тогда если ребро AB исключить из МОД, то дерево распадется на две связные компоненты, в одной из которых находится вершина C. Если C оказалась в одной компоненте с A, то вместо AB включаем в дерево ребро CB, которое короче AB. Аналогично, если C находится в одной компоненте с B, то меняем ребро AB на AC. В обоих случаях получаем дерево с длиной ребер меньшей, чем у исходного, что противоречит принадлежности AB к МОД. ■

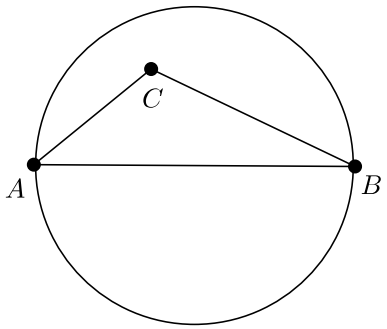


Рис. 10: Пояснение к лемме 4

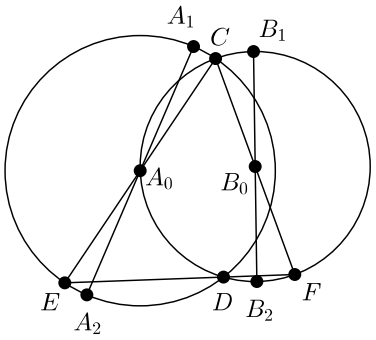


Рис. 11: Пояснение к лемме 5

Лемма 5. *Расстояние между центрами двух ребер МОД не меньше, чем половина длины каждого ребра.*

Доказательство. Предположим противное. Пусть A_1A_2 и B_1B_2 — ребра МОД с серединами A_0 и B_0 такие, что $|A_0B_0| \leq A_0A_2$ (рис. 11). Тогда их окружности влияния пересекаются в некоторых точках C и D. Не нарушая общности, предположим, что $|A_2B_2| \geq |A_1B_1|$. Из точки C проведем диаметры CE и CF кругов A_0 и B_0 соответственно. В $\triangle CDE$ угол $\angle CDE$ прямой, и в $\triangle CDF$ угол $\angle CDF$ тоже прямой, следовательно, EF проходит через точку D.

Из леммы 4 следует, что точка A_2 лежит на дуге $\widehat{ED}$, а точка B_2 – на дуге $\widehat{DF}$. Углы $\angle EA_2D$ и $\angle DB_2F$ тупые, поскольку дуги $\widehat{ED}$ и $\widehat{DF}$ меньше полуокружностей.

Следовательно, $|A_2B_2| < |A_2D| + |DB_2| < |ED| + |DF| = |EF| = 2|A_0B_0|$, т.е. $|A_2B_2| < |A_1A_2|$. Поскольку $|A_1B_1| \leq |A_2B_2|$, то и $|A_1B_1| < |A_2A_1|$. Покажем, что в этом случае A_1A_2 не может быть ребром МОД. Действительно, если удалить из покрывающего дерева ребро A_1A_2, то одна из точек A_1 или A_2 окажется в одной компоненте с ребром B_1B_2. Если это точка A_1, то заменяем A_1A_2 на A_2B_2 и получаем покрывающее дерево с меньшей длиной ребер. Если это точка A_2, то заменяем A_1A_2 на A_1B_1 с тем же результатом. Полученное противоречие доказывает утверждение леммы. ■

Из леммы 5 получаем, что центры окружностей влияния моста не могут лежать внутри окружностей влияния других мостов.

В *dumping* хранятся все дефектные ребра, и для каждого ребра хранится точка, нарушающая критерий леммы 2 для этого ребра. Из всех дефектных ребер, хранимых в *dumping* рассмотрим только те, которые являются мостами. Сайты, инцидентные мосту, называются *свободными*.

Лемма 6. *Для каждого свободного сайта может быть построено латентное ребро.*

Доказательство. Рассмотрим мост, инцидентный свободному сайту. По определению он является ребром МОД одной из исходных триангуляций и тогда, согласно лемме 4, внутри его окружности влияния нет сайтов одного с ним цвета. Но поскольку это ребро оказалось дефектным, внутрь этой окружности попали сайты противоположного цвета. Следовательно, для свободного сайта выполнено условие леммы 1, что и доказывает утверждение. ■

4.5.2 Построение последующих стартеров

Построение МОД для исходных триангуляций является трудоемкой задачей. Рассмотрим подробнее свойства моста. Лемма 4 утверждает, что окружность влияния моста является пустой окружностью триангуляции Делоне. Чтобы отказаться от использования МОД, вместе мостов из *dumping* будем использовать только такие ребра, окружность влияния которых была пустой в исходной триангуляции. Это легко проверить, взяв соседние точки в пучке исходной триангуляции и посчитав, попадают ли они в окружность влияния ребра. Такие дефектные ребра, имеющие пустой круг влияния, будем называть *псевдомостами*.

Пусть AC_1 — псевдомост (не ограничивая общности будем считать его черным). При удалении дефектного ребра AC_1 мы запомнили белую вершину, B, которая попала в его окружность влияния, что привело к удалению ребра из результирующей

триангуляции. Точка A была соединена латентным ребром с точкой B, а точка C_1 осталась свободной.

Тогда воспользовавшись алгоритмом 1, поиском в ширину, начиная с точки B найдем конец нового стартера. Приведем формальное описание функции построения последующих стартеров в алгоритме 5.

Алгоритм 5 Поиск последующих стартеров

```
1: function NEXTSTARTER()
2:     do
3:         if dumping == ∅ then
4:             return стартер не найден
5:         end if
6:         [edge, breaker] = dumping.get()
7:     while edge не является псевдомостом
8:     [fix, open] = edge
9:     starter_0 := open
10:    starte_1 := CALCULATESTARTER(open, fix, breaker)
11:    return [starter_0, starter_1]
12: end function
```

Таким образом, отказавшись от использования МОД пришли к простому и понятному способу поиска стартеров. По сравнению с изложением [3] удалось унифицировать процесс поиска первого и последующих стартеров.

4.6 Общая структура алгоритма

В предыдущих пунктах был подробно описан каждый шаг алгоритма слияния перекрывающихся триангуляции. Обобщим полученные результаты и напишем формальную процедуру слияния (алгоритм 6).

1. Первое латентное ребро существует (утв. 1), и оно является открытым ребром.

2. Существует алгоритм получения новых латентных ребер (утв. 2), и алгоритм, в результате которого образуются дефектные ребра.

3. Пока есть дефектные ребра, являющиеся псевдомостами, существуют стартеры (лем. 6), и существует способ их нахождения.

4. Если стартер найден, перейти к пункту 2, иначе закончить работу алгоритма

Алгоритм 6 Слияние триангуляций

```
1: procedure mergeTriangulations()
2:     starter := firstStarter()
3:     Добавить starter в триангуляцию
4:     constructLatent(starter)
5:     while dumping ≠ ∅ do
6:         starter := nextStarter
7:         if стартер не наден then
8:             return
9:         end if
10:        Добавить starter в триангуляцию
11:        constructLatent(starter)
12:    end while
13: end procedure
```

5 Оценка вычислительной сложности алгоритма

Построение исходных триангуляций Делоне относится к этапу предобработки, поэтому не учитывается при подсчете вычислительной сложности алгоритма. Алгоритм слияния перекрывающихся триангуляций Делоне можно разделить на три части:

- поиск стартеров,
- построение новых латентных ребер.

Рассмотрим вычислительную сложность алгоритма с использованием МОД. Построение МОД также относится к этапу предобработки. Однако известно, что сложность алгоритма Черитона-Тарьяна построения МОД составляет $O(n)$, что доказано в [4, стр.226-230].

5.1 Сложность поиска стартеров

Поиск первого стартера состоит из поиска самых левых точек множеств **B** и **W**, что имеет вычислительную сложность $O(n)$.

Поиск последующих стартеров состоит из двух этапов: первый заключается в выборе моста (осуществляется за $O(1)$, так как выбирается любой мост), второй предполагает перебор точек внутри окружности влияния моста.

Лемма 7. *Каждая точка может быть может быть накрыта не более, чем 6 окружностями влияния.*

Доказательство. Из лемм 4 и 5 следует, что окружности влияния имеют центры, не покрытые другими окружностями влияния. Рассмотрим точку A на плоскости. Пусть она покрыта двумя окружностями влияния O_1 и O_2. Тогда $|AO_1| \leq |O_1O_2|$ и $|AO_2| \leq |O_1O_2|$. В $\triangle AO_1O_2$ сторона $O_1O_2 > AO_1$ и $O_1O_2 > AO_2$, значит, $\angle O_1AO_2 \geq 60°$.

Если точка A покрыта k окружностями, то при попарных углах $O_iAO_{i+1} \geq 60°$ имеем $k \leq 6$. ■

По лемме 7 переборы по всем окружностям влияния имеют сложность $O(n)$. При отказе использования мостов и минимальных остовных деревьев для поиска стартеров нет оценки числа окружностей, попавших в круги влияния псевдомостов, однако эксперименты показывают вычислительную эффективность предложенного подхода.

5.2 Сложность построения латентных ребер

На этом шаге алгоритма разрушаются дефектные ребра, и строятся новые латентные ребра. Построение нового разноцветного ребра осуществляется на основе углового критерия (лемма 2) для пары конкурирующих ребер, т.е. занимает фиксированное время $O(1)$. Для каждого нового латентного ребра выполняется оценка корректности соседних с ним одноцветных ребер. Такая проверка дважды запускает угловой критерий. Если по результатам тестирования какое-то ребро уничтожается, то ставшее соседним другое одноцветное ребро вновь подвергается тесту. Таким образом, время на удаление одного одноцветного ребра также оценивается как постоянное $O(1)$. Всего ребер в исходных триангуляциях $O(n)$, значит сложность удаления одноцветных ребер $O(n)$.

5.3 Обобщение

К общему времени работы алгоритма добавляется время включения нового ребра в триангуляцию. При включении первого ребра, образуемого из стартера, может потребоваться полный перебор всех ребер, входящих в пучки двух инцидентных ему сайтов. Поэтому, общее время включения всех начальных ребер в пучки оценивается как $O(n)$. При этом включение очередного ребра уже не требует полного перебора ребер, поскольку новое ребро устанавливается в пучок вслед за текущим анализируемым ребром. Поэтому общее время на включение всех ребер есть $O(n)$.

Общее число построенных ребер не превосходит количество ребер в искомой триангуляции, то есть $O(n)$, а общее число разрушенных ребер не превосходит общее число ребер в исходных триангуляциях — тоже $O(n)$. Таким образом, мы доказали, что при использовании МОД вычислительная сложность предложенного алгоритма составляет $O(n)$. При отказе от использования МОД такой оценки, вообще говоря, доказано не было. Однако экспериментально установлена вычислительная эффективность предложенного алгоритма.

Размер памяти, используемой при работе алгоритма, также составляет $O(n)$.

6 Эксперименты

Предложенный алгоритм был реализован на языке программирования Python. Был разработан программный модуль, состоящий из информационных полей и методов. Поля содержат координаты исходных сайтов-точек, для каждого сайта храним список его сайтов-соседей, входящих в его пучок. А также содержит все описанные методы: поиск стартера по окружности и точке на ней, поиск первого стартера, удаление дефектных ребер, построение шва, поиск последующих стартеров и слияние триагуляций. Программу можно тестировать в двух режимах: ручной, при котором точки вводятся непосредственным указанием их места на плоскости, и автоматический, при котором исходные облака точек генерируются автоматически.

Эксперимент по проверке вычислительной сложности предложенного алгоритма проводился на данных разной размерности. Случайным образом были сгенерированы множества точек для двух триангуляций: $n_1 = n_2 = 500, 1000, \ldots, 5500$. На графике (рис. 12) видно, что асимптотика данного алгоритма составляет $O(n)$, однако в худшем случае алгоритм имеет сложность $O(n^2)$.

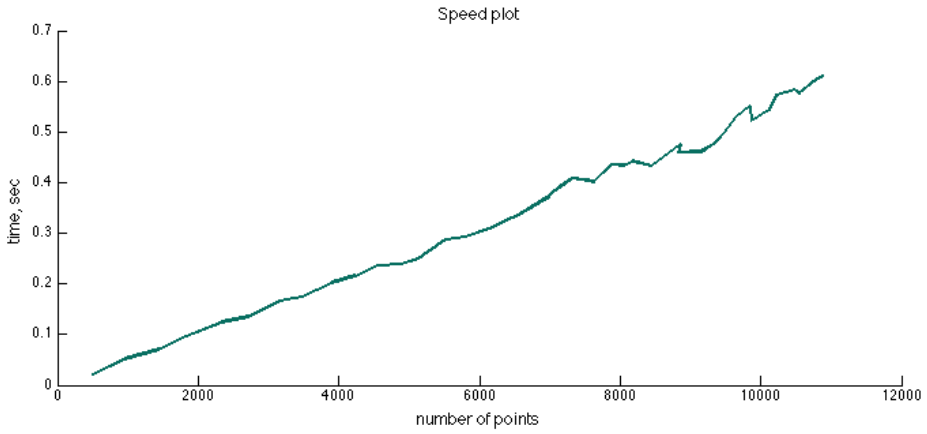


Рис. 12: Зависимость времени работы алгоритма от количества точек

В качестве входных данных используются начальные триангуляции (рис. 13), на выходе получаем объединенную триангуляцию (рис. 14).

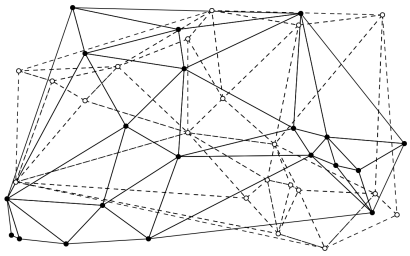


Рис. 13: Пример входных данных

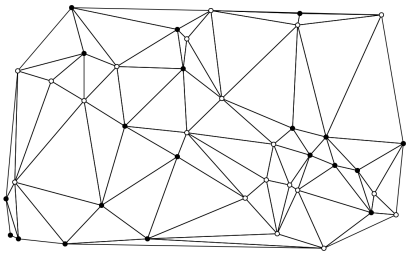


Рис. 14: Пример выходных данных

7 Заключение

В данной работе был представлен алгоритм слияния перекрывающихся триангуляций Делоне. Алгоритм получился проще с точки зрения использования структур данных, чем остальные известные подходы к решению данной задачи. Оценка вычислительной эффективности алгоритма была доказана для случая с использованием минимальных остовных деревьев, что предоставляет теоретический интерес. Удалось избавиться от использования МОД, однако в худшем случае вычислительная сложность алгоритма получается хуже, чем при использовании МОД. На практике же было проверено, что алгоритм является вычислительно эффективным. Таким образом, на защиту выносятся:

- существенное упрощение алгоритма по сравнению с существующими решениями,
- реализация алгоритма,
- экспериментально показана вычислительная эффективность алгоритма,
- на конференции GraphiCon2016 был представлен доклад на тему настоящей работы.

Список литературы

[1] D.G. Kirpatrick. Efficient computation of continuous skeletons // Proceedings of the 20th Annual IEEE Symposium on FOCS. 1979. С. 18–27.

[2] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedrons // SIAM J.Comput. vol.21. 1992. с. 671–696.

[3] Местецкий Л., Готман М. Метод слияния перекрывающихся триангуляций Делоне // GraphiCon2016 Труды Международной научной конференции. ННГАСУ, 2016. С. 289–294.

[4] Preparata Franco P., Shamos Michael I. Computational Geometry: An Introduction. New York, NY, USA: Springer-Verlag New York, Inc., 1985.

[5] D.T. Lee, B.J. Schachter. Two algorithms for constructing a Delaunay triangulation // International Journal of Computer and Information Science. vol. 9, no.3. 1980. С. 219–242.

[6] Л.М. Местецкий, Е.В. Царик. Слияние неразделенных триангуляций Делоне // Сложные системы: обработка информации, моделирование и оптимизация. Вып. 2. Тверь: Тверской гос.университет, 2004. С. 216–231.

[7] N. Dyshkant. Comparison of Point Clouds Acquired by 3D scanner // Lecture Notes in Computer Science: Discrete Geometry for Computer Imagery. vol.7749. 2013. С. 47–58.