Санкт-Петербургский государственный университет
Математико-механический факультет
Кафедра системного программирования

Проверка эквивалентности срединной и линейной осей многоугольника

Дипломная работа студента 545 группы

Подколзина Максима Валериевича

Научный руководитель:
к.ф.м.н., доцент ...../К. В. Вяткина/

Рецензент:
д.ф.м.н., профессор ...../О. Н. Граничин/

„Допустить к защите“
Заведующий кафедрой:
д.ф.м.н., профессор ...../А. Н. Терехов/

Санкт-Петербург
2008 г.

Содержание

1 Введение

В теории распознавания фигур широкое применение получило понятие срединной оси [1]. Срединная ось, состоящая из точек многоугольника, имеющих более одной ближайшей точки на границе многоугольника, является хорошим дескриптором многоугольника, но не очень удобна с точки зрения реализации в программе, потому что состоит из отрезков прямых линий и параболических дуг. Поскольку на практике работать с отрезками гораздо проще, чем с дугами параболы, срединную ось удобно приближать с помощью скелета, содержащего только прямолинейные отрезки, например, с помощью линейной оси.

Линейная ось, предложенная М. Танасе и Р. Велткампом в 2004 году [2], базируется на понятии прямолинейного скелета с использованием скрытых ребер при невыпуклых вершинах. Количество скрытых ребер может варьироваться, приводя к большему или меньшему сходству со срединной осью. Формально, для данного $\varepsilon > 0$ можно ввести понятие ε-эквивалентности между срединной осью и линейной осью.

Приближение срединной оси с помощью линейной может быть использовано для решения многих геометрических задач. Важную роль при этом играет параметр ε, который определяет точность аппроксимации, и, следовательно, влияет на качество конечного результата. В разных ситуациях качественный результат может достигаться на сильно отличающихся ε, как следствие задать в программе единое значение ε для всех входных данных невозможно. Одно из возможных решений – это построение нескольких результатов для различных ε, и выбор подходящего, возможно, с подсказками со стороны человека. В том случае, если результат с точки зрения визуального восприятия неудовлетворительный, программа получает инструкцию уменьшить ε и построить изображение заново, что требует перестраивания всех объектов, в том числе линейных осей. Однако реально большую часть линейных осей перестраивать не нужно, потому что оси, построенные для старого значения ε, часто остаются эквивалентны срединной оси и для нового ε. До сих пор оптимизировать переход при изменении ε было невозможно, так как не было эффективного метода проверки эквивалентности.

В качестве конкретного примера можно рассмотреть задачу восстановления поверхности по набору срезов. Задача ставится следующим образом: дано множество срезов некоторой поверхности в $\mathbb{R}^3$ горизонтальными плоскостями. Требуется построить триангулированную кусочно-линейную поверхность, близкую к исходной.

Одно из решений этой задачи требует построения линейных осей как промежуточных объектов при триангуляции, причем число линейных осей по порядку величины может быть кубическим относительно числа срезов. Но наличие в многоугольниках-контурах вершин с внутренним углом, близким к 2π, может стать причиной ухудшения качества поверхности. В этой задаче с использованием алгоритма проверки эквивалентности можно было бы существенно оптимизировать переход к другому ε, если перестраивать только те линейные оси, которые окажутся неэквивалентными с новым ε.

Понятие ε-эквивалентности – не единственный подход для оценки сходства срединной и линейной осей. Цели данной работы – провести исследование различных типов эквивалентностей осей и разработать алгоритм проверки эквивалентности для данного ε, данного многоугольника, данных линейной и срединной осей.

Мы введем определение сильной эквивалентности, покажем, что результаты, доказанные ранее для обычной ε-эквивалентности, справедливы и для нового определения, и предложим эффективный алгоритм проверки сильной эквивалентности. Также мы рассмотрим понятие геометрической эквивалентности, классифицируем данный тип эквивалентности относительно других типов, и опишем алгоритм его проверки.

2 Основные определения и постановка задачи

Для начала, дадим определения базовым понятиям. Большая часть терминологии взята из [3].

Определение 2.0.1. Пусть P - произвольный многоугольник. *Срединной осью* $M(P)$ назовем множество центров кругов, целиком лежащих внутри P и касающихся границы P по крайней мере в двух точках.

Иначе: это множество точек, которые имеют больше одной ближайшей точки на границе P. Соответствующие ближайшие стороны или вершины P называются *сайтами*, порождающими данную точку.

Определение 2.0.2. Области, на которые $M(P)$ разбивает многоугольник, называются *ячейками* срединной оси.

С процессом построения срединной оси тесно связано понятие равномерного фронта.

Определение 2.0.3. *Равномерным фронтом* $U(d)$ для многоугольника P и расстояния d является множество точек внутри P, находящихся на расстоянии d от границы P.

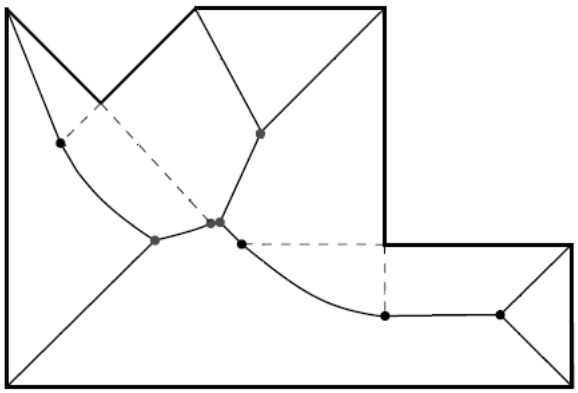

Рис. 1. Пример срединной оси многоугольника.

В общем случае, срединная ось состоит из отрезков прямых линий и параболических дуг (рис. 1). Причиной появления параболических дуг является тот факт, что геометрическое место точек, равноудаленных от данной точки и данной прямой - парабола. В нашем случае точкой является невыпуклая вершина многоугольника, а отрезок прямой - сторона многоугольника. Линейная ось, в отличие от срединной оси, состоит только из прямолинейных отрезков.

Определение 2.0.4. Пусть $\{v_1, ..., v_m\}$ - невыпуклые вершины многоугольника P, пусть $k = (k_1, ..., k_m)$ - набор неотрицательных целых чисел. Заменим каждую вершину v_i $k_i + 1$ совпадающей вершиной, соединенными k_i ребрами нулевой длины. Выберем направления ребер таким образом, чтобы внутренние углы при каждой из $k_i + 1$ вершин были одинаковыми. Обозначим полученный многоугольник P^k. *Линейной осью* $L^k(P)$, соответствующей набору k, назовем граф, ребра которого вычерчиваются выпуклыми вершинами при распространении линейного фронта[1] P^k.

Дополнительные ребра, добавленные в невыпуклые вершины, называются *скрытыми*.

На рис. 2 изображена линейная ось для набора $k = (1, 1)$. Обратим внимание, что след выпуклых вершин при распространении линейного фронта входит в линейную ось, а след скрытых вершин (изображен пунктиром) - нет. Также в линейную ось не входит след невыпуклых вершин многоугольника и линейного фронта.

[1] Процесс распространения линейного фронта - это процесс, при котором все ребра многоугольника движутся внутрь с постоянными скоростями, оставаясь при этом параллельными себе.

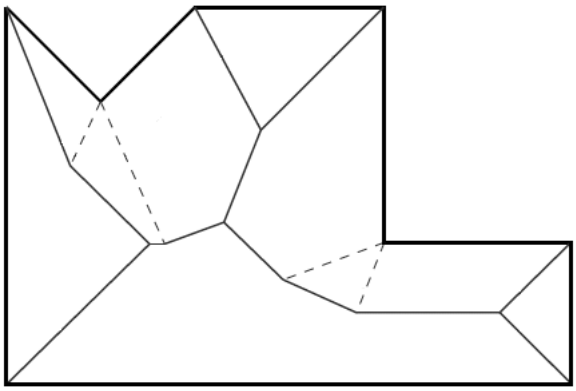

Рис. 2. Пример линейной оси многоугольника.

Добавим, что линейная ось не обязательно связна, если по крайней мере два из внутренних углов многоугольника больше $\frac{3\pi}{2}$, и в них добавлено 0 скрытых ребер. Для того, чтобы не допускать такой ситуации, мы всегда будем полагать, что при невыпуклых вершинах с углом больше $\frac{3\pi}{2}$ есть по крайней мере одно скрытое ребро. В таких условиях линейная ось всегда связна [2].

С увеличением числа скрытых ребер линейная ось все лучше приближает срединную ось. Введем строгое понятие ε-эквивалентности между линейной и срединной осями, для чего нам понадобится ряд дополнительных определений.

Определение 2.0.5. *Геометрическим графом* G называется пара множеств (V, E), где V - конечное множество точек в $\mathbb{R}^2$, E - конечное множество непересекающихся во внутренних точках простых кривых. Элементы множества V называются *вершинами*, элементы E - *ребрами* графа. Каждое ребро должно соединять две вершины.

Любой срединной оси, как и линейной, можно сопоставить геометрический граф этой оси: множество вершин V - это вершины оси либо степени 1 (они соответствуют выпуклым вершинам многоугольника), либо степени ≥ 3, множество ребер E - непрерывные кусочно-гладкие кривые, их соединяющие. Для удобства изложения мы будем отождествлять выпуклые вершины многоугольника и соответствующие им вершины графа, хотя формально это разные объекты.

Для фиксированного многоугольника P обозначим через $G_M = (V_M, E_M)$ геометрический граф срединной оси, $G_{L^k} = (V_{L^k}, E_{L^k})$ - геометрический граф линейной оси. В данной работе мы будем рассматривать только такие графы, вершины которых имеют степень 1 или 3. В случае когда вершина будет иметь степень $d > 3$, будем ее рассматривать как $d - 2$ совпадающие вершины степени 3, соединенные ребрами таким образом, чтобы граф, индуцированный этими вершинами, был деревом. Такая трактовка очень удобна, так как в этом случае $|V_M| = |V_{L^k}|$ [5].

В следующих определениях ε – произвольное вещественное число больше 0.

Определение 2.0.6. Пусть u и v - вершины срединной оси, соединенные ребром e, пусть S_1, S_2, S_3 и S_1, S_2 и S_3' - порождающие их сайты. Ребро e называется ε-*коротким* тогда и только тогда, когда $d(u, S_3') < (1 + \varepsilon)d(u, S_3)$ или $d(v, S_3) < (1 + \varepsilon)d(u, S_3')$.

Определение 2.0.7. Ребро, не являющееся ε-коротким, называется ε-*длинным*.

Определение 2.0.8. ε-*кластером* $C(v)$ назовем множество вершин, достижимых из v по ε-коротким ребрам.

Введенные определения расширяются естественным образом на геометрический граф срединной оси. Именно, назовем ребро $e \in E_M$ ε-коротким, если составляющие e ребра срединной оси являются

ε-короткими. Определения ε-длинного ребра и ε-кластера G_M аналогичны 2.0.7 и 2.0.8. В дальнейшем, мы будем не оговариваясь использовать понятия $C(v)$ для $v \in V_M$ и ε-длинного или ε-короткого ребра e для $e \in E_M$.

Таким образом, все вершины V_M разбиваются на классы эквивалентности, которые мы называем ε-кластерами. Для данного многоугольника таких разбиений может быть много, в зависимости от ε. Пример геометрического графа срединной оси с одним из вариантов разбиения на ε-кластеры изображен на рис. 3.

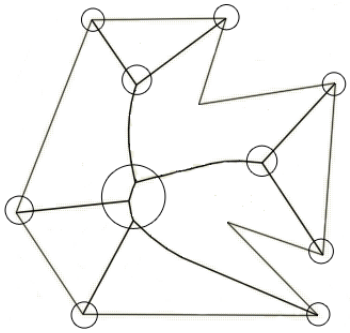

Рис. 3. Пример геометрического графа срединной оси и разбиения на ε-кластеры.

Обратим внимание, что для выпуклых вершин p соответствующий ε-кластер всегда состоит из одной p.

Определение 2.0.9. Будем говорить, что срединная ось $M(P)$ *ε-эквивалентна* линейной оси $L^k(P)$, если существует биекция $f : V_M \to V_{L^k}$, такая что

1. $f(p) = p$ для всех выпуклых вершин p;
2. для любых $u, v \in V_M$, таких что $C(u) \neq C(v)$, существует ребро в E_M, соединяющее u и v, тогда и только тогда, когда существует ребро в E_{L^k}, соединяющее $f(u')$ и $f(v')$, где $u' \in C(u)$, $v' \in C(v)$.

По смыслу это означает, что при склеивании вершин каждого ε-кластера в одну, а также аналогичном склеивании вершин V_{L^k}, мы получаем изоморфные графы. Очевидно, что изоморфизм исходных геометрических графов является более сильным свойством, из которого автоматически следует ε-эквивалентность, но на практике это свойство не всегда выполняется. В этом случае проверка ε-эквивалентности сводится к поиску подходящего отображения между геометрическими графами, что в общем случае является непростой задачей (она может оказаться NP-трудной задачей).

Тривиальный алгоритм перебора всех отображений, очевидно, является экпоненциальным по времени, так как общее число отображений между графами с n вершинами равно $n!$. Даже если учесть, что для выпуклых вершин перебор не требуется, число оставшихся вершин $k = O(n)$ ($k = \frac{n-2}{2}$, если быть точным), следовательно, число отображений все равно не является полиномиальным.

Поскольку ε-эквивалентность – не единственный возможный способ оценки степени сходства осей, представляет интерес рассмотрение других подходов, для которых проверка может быть выполнена за полиномиальное время. В данной работе мы ставим задачу исследования различных типов эквивалентности, их классификации, а также разработки эффективных алгоритмов проверки различных типов эквивалентности, применимых к широкому классу многоугольников.

Нас будут интересовать простые многоугольники. Поэтому здесь же упомянем еще один полезный факт, связанный с ними.

Теорема 2.0.10. *Если P - простой многоугольник, то G_M и G_{L^k} - деревья.*

3 Пример многоугольника и линейной оси, не являющейся ε-эквивалентной срединной оси

В этом параграфе мы приведем пример ситуации, при которой линейная и срединная оси не ε-эквивалентны.

Мы будем рассматривать $\varepsilon \in (0, \frac{2}{\sqrt{3}} - 1)$, в дальнейшем покажем, почему верхняя граница на ε именно такая. Наша задача – построить многоугольник и указать линейную ось, которая была бы не ε-эквивалентна срединной оси.

Решение будем искать среди пятиугольников, симметричных относительно вертикальной оси, один из внутренних углов которого невыпуклый, а остальные - меньше $\frac{\pi}{2}$. Пример такого многоугольника изображен на рис. 4.

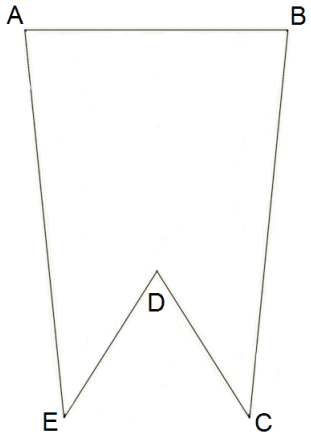

Рис. 4. Симметричный многоугольник, угол при вершине D невыпуклый.

В качестве k возьмем набор (2), то есть поместим в вершину D два скрытых ребра. На рис. 5 показаны срединная ось M (а) и линейная ось L^k (б).

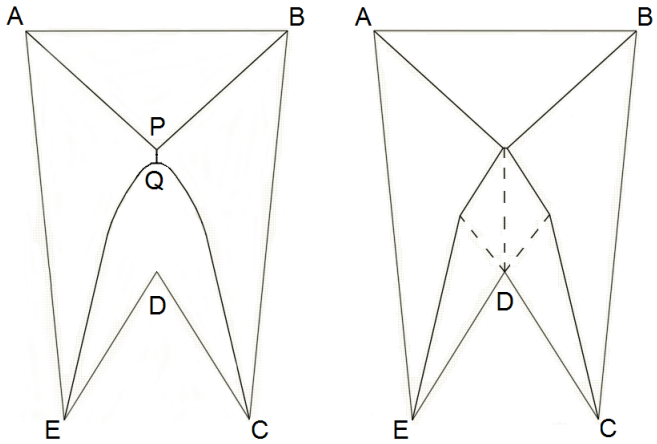

Рис. 5. (а) Срединная ось M и (б) линейная ось L^k при $k = (2)$.

Введем следующие обозначения: обозначим через P и Q вершины G_M, находящиеся внутри многоугольника; расстояние от P до AB обозначим через r_1, очевидно, расстояния от P до AE и BC будут также равны r_1; $|QD|$ обозначим через r_2, очевидно, расстояния от Q до AE и BC

будут также равны r_2; расстояние между P и Q обозначим a; внешний угол при вершине D через ϕ. Также обозначим внутренний угол при вершине A через α. Так как он меньше $\frac{\pi}{2}$, $\alpha = \frac{\pi}{2} - \delta$, где $\delta = \frac{\pi}{2} - \alpha > 0$.

Для начала предположим, что многоугольник построен, и проанализируем срединную ось M.

Первое замечание: для того чтобы вершины срединной оси P и Q попали в разные ε-кластеры, достаточно, чтобы $\frac{r_2+a}{r_1} \geq 1+\varepsilon$. Действительно, по определению это условие эквивалентно выполнению двух неравенств: $\frac{r_2+a}{r_1} \geq 1+\varepsilon$ и $\frac{r_1+a}{r2} \geq 1+\varepsilon$, но второе автоматически следует из первого, потому что $r_1 > r_2$.

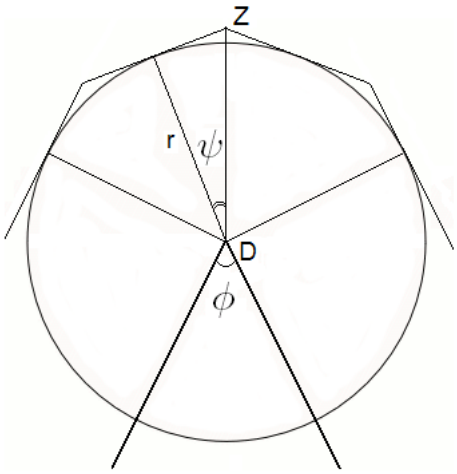

Рис. 6. Распространение линейного и равномерного фронтов вблизи вершины D.

Далее, рассмотрим насколько медленнее распространяется равномерный фронт, чем самая быстрая из точек в линейном фронте. Эта точка всегда является одной из скрытых вершин, обозначим ее через Z (рис. 6). В момент времени r $\frac{r}{|DZ|} = cos\psi$, а $\psi = \frac{\pi-\phi}{6}$. Таким образом, $|DZ| = \frac{r}{cos(\frac{\pi-\phi}{6})}$.

Для достижения нашей цели, необходимо выполнение следующих условий: чтобы точки P и Q попали в разные кластеры, для чего достаточно $\frac{r_2+a}{r_1} \geq 1+\varepsilon$, и чтобы линейный фронт, а именно точка Z, достигла точки P до того, как ее достигнет равномерный фронт. Последнее означает, что $\frac{r_1}{cos(\frac{\pi-\phi}{6})} > a + r_2$. В нашей власти менять ϕ произвольно от 0 до π, а наибольшего значения эта дробь достигает при $\phi = 0$, косинус при этом равен $\frac{\sqrt{3}}{2}$. Не теряя общности положим $r_1 = 1$, и тогда это условие будет гарантированно выполнено, если $a < \frac{1}{cos(\frac{\pi-\phi}{6})} - 1$.

Собственно, построение: выберем a - произвольное число от ε до $\frac{2}{\sqrt{3}} - 1$. Далее, можно установить угол при вершине A, и соответственно при вершине B, настолько большим, чтобы в точке Q расстояние до сторон r_2 было бы близко к r_1. Строго говоря, нам нужно, чтобы $r_2 \geq 1 + \varepsilon - a$. Это возможно, потому что правая часть меньше 1.

Итак, построим отрезок AB и отложим от вершин A и B лучи под выбранными углами. Пусть P - центр окружности, касающейся лучей и AB. Отметим вершину Q на расстоянии a от P и вершину D на расстоянии r_2 от Q. Заметим, что у нас еще свобода в расстановке вершин C и E, что означает, что мы можем установить любой сколь угодно малый угол ϕ. Выберем его так, чтобы $a < \frac{1}{cos(\frac{\pi-\phi}{6})} - 1$, то есть выберем $\phi < \pi - 6 \cdot arccos\frac{1}{1+a}$. Пятиугольник $ABCDE$ построен, вершины P и Q попадают в разные ε-кластеры.

Мы имеем два возможных отображения между вершинами геометрических графов M и L^k, и, очевидно, ни одно из них не удовлетворяет условию ε-эквивалентности.

4 Понятие сильной эквивалентности

В этом параграфе мы усилим ε-эквивалентность дополнительным условием на биекцию f и выделим основные свойства нового определения.

Определение 4.0.11. Будем говорить, что между срединной осью $M(P)$ и линейной осью $L^k(P)$ есть *сильная эквивалентность*, если существует биекция $f : V_M \to V_{L^k}$, удовлетворяющая условиям 1 - 2 определения 2.0.9 и

3. Для любого ε-кластера C подграф G_{L^k}, состоящий из вершин $f(C)$ и ребер между ними, связен.

Заметим, что в случае простого многоугольника P, дополнительное условие можно сформулировать иначе:

Утверждение 4.0.12. *Пусть P - простой многоугольник, и для $f : V_M \to V_{L^k}$ верны условия 1 - 2 определения 2.0.9. Тогда $3 \Leftrightarrow 3'$:*

$3'$. *Для любых двух ε-кластеров C_1 и C_2 между $f(C_1)$ и $f(C_2)$ существует не более одного ребра.*

Доказательство. Докажем каждое из следствий.

- $3 \Rightarrow 3'$. Предположим, что это не так, и между $f(C_1)$ и $f(C_2)$ есть по крайней мере два ребра: $u_i \in f(C_1), u_i' \in f(C_2), (u_i, u_i') \in E_{L^k}$ для $i = 1, 2$. Но в силу связности графов, индуцированных $f(C_1)$ и $f(C_2)$, между u_1 и u_2 есть путь в $f(C_1)$, а между u_1' и u_2' - в $f(C_2)$. Полученный цикл противоречит тому, что G_{L^k} - дерево (см. теорему 2.0.10).
- $3' \Rightarrow 3$. Из любого ε-кластера C, такого что $|C| = l$, не соответствующего выпуклой вершине, выходит ровно $l+2$ ε-длинных ребра, причем ε-кластеры, в которые они ведут, все различны. Предположим, что граф, индуцированный $f(C)$, состоит по крайней мере из двух компонент связности. Тогда в графе G_{L^k} от вершин $f(C)$ исходит по крайней мере $l+4$ ребра, ведущих в вершины, которым соответствуют вершины G_M, входящие в ε-кластеры, отличные от C. Так как эти ε-кластеры должны быть различны, получаем противоречие тому, что оси ε-эквивалентны.

□

В работах [3], [5] были доказаны теоремы о достаточных условиях ε-эквивалентности. Можно показать, что для отображений f, которые предлагались в доказательствах, условие 3 имеет место[2]. Следовательно, можно сформулировать аналогичные теоремы применительно к сильной эквивалентности, а также сохранят силу и алгоритмы нахождения набора k, достаточного для эквивалентности. Однако вопрос, равносильны ли эти два определения, является открытым.

[2][3], стр. 84. Множество вершин, соответствующих ε-кластеру, определяется как класс эквивалентности по отношению $b_i \simeq b_j \Leftrightarrow$ любое ребро на пути из b_i в b_j удовлетворяет определенному условию. Следовательно, этот класс эквивалентности связен в L^k.

[5], стр. 129. Связность двойственного кластера напрямую следует из Леммы 14.

5 Понятие геометрической эквивалентности

Ниже мы предложим еще один тип эквивалентности осей, основанный на геометрическом расположении вершин G_M и G_{L^k}. Мы дадим формальное определение геометрической эквивалентности, а также классифицируем этот тип эквивалентности относительно сильной и ε-эквивалентности. Тем самым мы обоснуем иерархию рассмотренных типов эквивалентности.

Прежде всего, мы введем еще несколько определений, заимствованных из [4].

Определение 5.0.13. Пусть $VC(S_1)$ и $VC(S_2)$ - ячейки срединной оси, соответствующие сайтам S_1 и S_2, (u, v) - ребро, образованное пересечением границ $VC(S_1)$ и $VC(S_2)$. *Барьером* для ребра (u, v) будем называть пару отрезков, соединяющих некоторую точку b на этом ребре с ближайшими точками на S_1 и S_2. Точку b будем называть центром барьера (рис. 7а)[3].

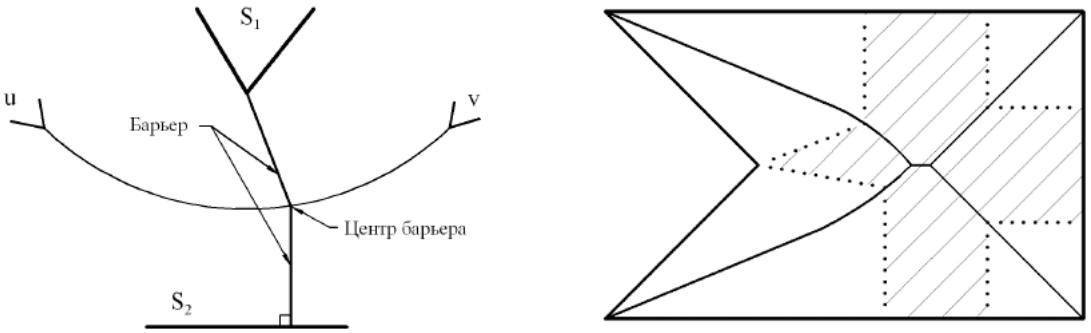

Рис. 7. (а) Барьер для ребра (u, v), (б) Преграда для ε-кластера из двух вершин.

Определение 5.0.14. Преградой для ε-кластера C назовем множество барьеров для всех длинных ребер, имеющих один из своих концов внутри C, а другой вне C (рис. 7б).

Обозначим через $P(C)$ область многоугольника, ограниченную преградой и содержащую кластер C.

Утверждение 5.0.15. *$P(C)$ – связное множество.*

Определение 5.0.16. Назовем срединную и линейную оси *геометрически эквивалентными*, если найдется набор барьеров для ε-длинных ребер $M(P)$ такой, что каждому ε-кластеру $C = \{v_1, ..., v_l\} \subset V_M$ можно сопоставить множество вершин $\{v'_1, ..., v'_l\} \subset V_{L^k}$, так что все вершины $\{v'_1, ..., v'_l\}$ и соединяющие их ребра будут находиться внутри преграды для кластера C.

Аналогичный объект уже фактически использовался для доказательства ε-эквивалентности [5], однако не выделялся в самостоятельное определение. Нам же в рамках исследования различных подходов к оценке сходства осей удобно рассматривать и его как один из возможных типов эквивалентности.

Как уже было отмечено, из существования геометрической эквивалентности следует ε-эквивалентность, и можно показать, что следует также сильная эквивалентность [5]. Таким образом, среди всех рассмотренных в работе подходов самой сильной эквивалентностью является геометрическая, далее сильная эквивалентность и ε-эквивалентность. Для первых двух типов ниже представлены алгоритмы, позволяющие по входным графам срединной и линейной осей ответить, являются ли они эквивалентны.

[3] Рисунки барьера и преграды также взяты из [4].

6 Алгоритм проверки сильной эквивалентности срединной и линейной осей

В этом параграфе мы представим алгоритм проверки сильной эквивалентности срединной оси $M(P)$ и линейной оси $L^k(P)$ для данного k, $\varepsilon > 0$ и простого многоугольника P.

Мы будем рассматривать только геометрические графы G_M и G_{L^k}, а не соответствующие оси, причем, как было упомянуто выше, нам удобнее трактовать каждую вершину степени $d \geq 4$ как $(d-2)$ совпадающие вершины степени 3. Т.е. будем полагать, что на вход алгоритму вместе с P и ε даются геометрические графы G_M и G_{L^k}. Иначе, если даны срединная ось и линейная ось, сразу построим G_M и G_{L^k} за линейное время и будем работать с ними.

Замечание 6.0.17. Задача, рассматриваемая в этом разделе, может быть сформулирована в других терминах как частный случай задачи о совместимости филогенических деревьев [6]: для данных m филогенических деревьев T_1, ..., T_m требуется установить, существует ли такое дерево T, из которого можно получить каждое дерево T_i за счет удаления ребер. Доказано, что в общем случае существует решение этой задачи за время $O(nm)$. Предложенный нами алгоритм отличается от алгоритма для решения общей задачи, так как в значительной степени использует тот факт, что геометрический граф линейной оси – кубическое дерево, при этом не уступает в эффективности.

Теорема 6.0.18. *Для любого ε и простого многоугольника P можно за линейное время и с использованием линейной памяти установить, являются ли $M(P)$ и $L^k(P)$ сильно эквивалентными.*

Доказательство. Сначала опишем алгоритм проверки сильной эквивалентности для данного ε и данных геометрических графов G_M и G_{L^k}, затем докажем его корректность и проанализируем время работы и затрачиваемую память.

В реализации воспользуемся объектно-ориентированным подходом. Именно, предположим, что есть тип Vertex, который имеет поле visited (хранит, была ли вершина посещена), поле cluster (ссылка на кластер, содержащий эту вершину), необходимые поля для хранения инцидентных ребер. Заведем также тип Edge - ребро в графе, объекты Edge будут также иметь поле visited. Заведем тип Cluster - некоторое множество в графе вершин. Для графа G_M экземпляр Cluster будет хранить ε-кластеры, для графа G_{L^k} - образы ε-кластеров при биекции (двойственные кластеры). Также мы будем использовать такие структуры данных:

- Очередь Q для хранения пар вершин (v, v'), где $v \in V_M$, $v' \in V_{L^k}$.
- Итоговое отображение f. Для удобства, мы определим его не на множествах вершин, а на множествах кластеров, т.е. если C - ε-кластер, то $f(C)$ - соответствующий ему двойственный кластер (в программе будем использовать обозначение `f[C]`). В конце алгоритма по f можно легко восстановить отображение на вершинах F: для каждого ε-кластера C возьмем произвольную перестановку вершин $f(C)$ относительно C, и определим F на элементах C согласно этой перестановке.

Изначально очередь Q пуста, все вершины и ребра непосещены, $f \equiv null$, для вершин $v \in V_M$ присвоим $v.cluster \leftarrow C(v)$, для вершин $v' \in V_{L^k}$ присвоим $v'.cluster \leftarrow null$.
Алгоритм будет состоять из нескольких процедур:

1. *Основная функция*(G_M, G_{L^k}) - производит обход в ширину двух графов и возвращает ответ.
2. *Перейти*(v, v', u, u') - осуществляет переход из вершины v в u в графе G_M и одновременно переход из v' в u' в графе G_{L^k}; также процедура проверяет, нарушаются ли при переходе условия, накладываемые на f, и в случае нарушения завершает алгоритм с ответом "нет".

3. *Расширить*(u) - начиная от u, проходит по вершинам графа, пока не окажется в непосещенной вершине;

Теперь напишем весь алгоритм на псевдокоде.

Вход: графы G_M и G_{L^k}, набор ε-кластеров G_M.

Выход: "нет", если оси не являются сильно эквивалентными; "да", если сильно эквивалентны и биекция, удовлетворяющая условиям определения 4.0.11.

Основная функция (G_M, G_{L^k}) :

```
1.  /* Инициализация: добавить выпуклые вершины */
2.  for (p1, p2 - соответствуют выпуклым вершинам в G_M и G_{L^k}):
3.    пометить p1, p2 как посещенные
4.    f[p1.cluster] ← { p2 }
5.    Q.push(p1, p2)  6.  /* Обход в ширину */
7.  while (Q.size() > 1):
8.    (v, v') ← Q.poll()
9.    /* инвариант: v и v' будут инцидентны ровно одному непосещенному ребру */
10.   u ← вершина G_M, такая что ребро (v, u) не посещено
11.   u' ← вершина G_{L^k}, такая что ребро (v', u') не посещено
12.   Перейти(v, v', u, u')
13. return f
```

Перейти(v, v', u, u') :

```
1.  пометить ребра (v, u), (v', u') как посещенные
2.  if (u'.visited):
3.    /* При невыполнении условия: ответ "нет"*/
4.    CHECK(f[u.cluster] = u'.cluster)
5.  else:
6.    if (f[u.cluster] = null):
7.      /* Создаем новый двойственный кластер, состоящий из u' */
8.      u'.cluster ← { u' }
9.      f[u.cluster] ← u'.cluster
10.   else:
11.     /* Иначе добавляем u' в уже существующий двойственный кластер */
12.     u'.cluster ← f[u.cluster]
13. /* В этот момент равновесие между кластерами может быть нарушено:
14.    число посещенных вершин в u.cluster может быть не равно размеру f[u.cluster] */
15. if (u.visited and not u.cluster.allvisited):
16.   Расширить(u)
17. if (u'.visited and not u'.cluster.allvisited):
18.   u'.cluster.add(Расширить(u'))
19. пометить u и u' как посещенные
20. if (число помеченных ребер, инцидентных u.cluster, равно u.cluster.size + 1):
21.   /* Кластер u.cluster обработан - добавляем очередную вершину в Q */
22.   пометить все внутренние ребра в u.cluster как посещенные
23.   пометить все внутренние ребра в u'.cluster как посещенные
24.   w ← вершина u.cluster, из которой выходит одно непосещенное ребро
25.   w' ← вершина u'.cluster, из которой выходит одно непосещенное ребро
26.   Q.push(w, w')
```

Расширить(u) :

```
1.  /* u будет инцидентно ровно одно непосещенное ребро */
2.  w ← вершина, такая что ребро (u, w) не посещено
3.  пометить ребро (u, w) как посещенное
4.  if (not w.visited):
5.    пометить w как посещенную
6.    return w
7.  return Расширить(w)
```

Описание алгоритма закончено. Для доказательства его корректности нам понадобится ряд вспомогательных утверждений. Сначала докажем инварианты, которые использовались в алгоритме.

Лемма 6.0.19. *Пусть u - любая вершина G_M, не совпадающая с выпуклой вершиной. Тогда в тот момент, когда число помеченных ε-длинных ребер, инцидентных $C(u)$, станет равно $|C(u)|+1$, все вершины $C(u)$ будут посещены.*

Доказательство. Разобъем вершины $C(u)$ на три группы: в i-ю группу включим такие вершины, которым инцидентно i посещенных ε-длинных ребра. Обозначим количества этих вершин k_0 , k_1 и k_2 соответственно. Тогда $l = k_0 + k_1 + k_2$, где $l = C(u)$ – размер кластера. С другой стороны, можно посчитать число помеченных ребер, входящих в $C(u)$: $l + 1 = 0 \cdot k_0 + 1 \cdot k_1 + 2 \cdot k_2$. Отсюда получаем, что $k_2 = k_0 + 1$.

Имеем: во-первых, будут посещены $k_1 + k_2$ вершин, находящихся "на границе" $C(u)$. Во-вторых, $k_2 - 1$ раз будет вызвана функция *Расширить*, в результате которой будет обработано еще $k_0 = k_2 - 1$ вершин $C(u)$. В этот момент станут посещены все l вершин. □

Тем самым мы оправдали строки 20-21 функции *Перейти*. Следующая лемма доказывает инвариант, на который мы опирались в строках 9-11 *Основной функции*:

Лемма 6.0.20. *В Q попадают только такие пары вершин (v, v'), кроме последней, что из ε-кластера $C(v)$ выходит ровно одно непосещенное ребро в G_M, и из двойственного кластера $f(C(v))$ выходит ровно одно непосещенное ребро в G_{L^k}.*

Доказательство. Добавление пар в очередь осуществляется только в двух местах алгоритма - в инициализации и в конце функции *Перейти*. В первом случае условие очевидно, проверим второй.

Для краткости рассмотрим ε-кластер $C(u)$, рассуждения для $f(C(u))$ аналогичны. Как следует из предыдущей Леммы, если выполнено условие строки 20 *Перейти*, все вершины кластера обработаны. Рассмотрим все ребра, исходящие из вершин $C(u)$. Среди них $|C(u)| + 2$ – ε-длинных, все остальные не выходят за пределы кластера. Следовательно, если пометить все внутренние ребра в $C(u)$, непосещенным останется ровно одно ε-длинное ребро.

Заметим здесь же, что эти рассуждения доказывают корректность строк 24-25. □

Лемма 6.0.21. *После каждого вызова функции* Расширить *имеет место инвариант: каждой посещенной вершине ε-кластера инцидентно либо 1, либо 3 помеченных ребра.*

Доказательство. В тот момент, когда вершина становится посещена, ей, очевидно, инцидентно ровно одно посещенное ребро - то, по которому мы пришли. В тот момент, когда мы приходим в нее повторно, вызывается функция *Расширить* (рис. 8).

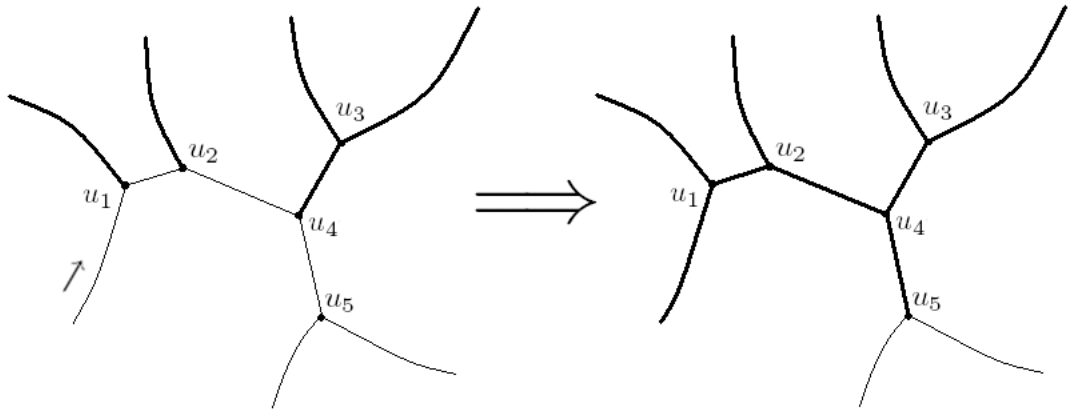

Рис. 8. Пример вызова функции *Расширить*. Вершины $u_1, ..., u_5$ лежат в одном ε-кластере, посещенные ребра выделены жирными линиями. В результате вызова *Расширить*(u_1) будут посещены ребра (u_1, u_2), (u_2, u_4), (u_4, u_5), и возвращена вершина u_5.

Следовательно, в момент вызова *Расширить*(u) вершине u инцидентно два помеченных ребра. После этого помечается третье ребро (u, w), и либо вершине w инцидентно одно помеченное ребро, в этом случае она возвращается, и либо два, и ситуация повторяется снова. Инвариант сохраняется. □

Лемма 6.0.22. *После каждого вызова функции* Расширить *каждой вершине в двойственного кластере $f(C)$ инцидентно либо 1, либо 3 помеченных ребра.*

Доказательство. Аналогично предыдущей лемме. □

Лемма 6.0.23. *При вызове функции* Расширить(u)*, вершине u инцидентно ровно 1 непосещенное ребро.*

Доказательство. Функция *Расширить*(u) вызывается только тогда, когда мы приходим в u повторно (строки 15-18 *Перейти*), значит, до вызова функции *Перейти* все три инцидентных ребра не могли быть посещены. Как показано в двух предыдущих леммах, вершине не может быть инцидентно два посещенных ребра. Следовательно, было посещено одно ребро, и перед вызовом *Расширить* их станет два. Соответственно, одно ребро будет непосещено. □

Следовательно, инвариант в строке 1 *Расширить* выполнен.

Мы доказали, что алгоритм будет работать так, как предписано – все необходимые инварианты имеют место. Однако все еще не ясно, почему корректен возвращаемый результат. Для начала покажем, что в процессе обхода мы посетим все вершины, или алгоритм закончится с ответом "нет".

Лемма 6.0.24. *После каждого вызова функции* Перейти *имеет место инвариант: число посещенных вершин в кластере C равно числу вершин в $f(C)$.*

Доказательство. В начале алгоритма инвариант, очевидно, выполнен, так как посещены только выпуклые вершины. Рассмотрим вызов *Перейти*(u, v, u', v') и разберем 3 случая:

1. обе вершины u и u' не посещены. Тогда инвариант выполнен, так как u становится посещена, а u' начинает образовывать новый двойственный кластер, соответствующий $C(u)$ (строчки 8-9);

2. одна из вершин u и u' посещена, а другая нет. Перед вызовом функции *Расширить* равенство может быть нарушено, так как мы повторно пришли в одну из вершин. Однако после этого равенство восстанавливается, так как функция *Расширить* находит новую вершину;

3. случай, когда обе вершины посещены, аналогичен предыдущему пункту.

□

Следствие 6.0.25. *После каждого вызова функции* Перейти *число посещенных вершин в G_M равно числу посещенных вершин в G_{L^k}.*

Лемма 6.0.26. *При обходе будут посещены все вершины обоих графов, либо алгоритм остановится с ответом "нет".*

Доказательство. Обход начинается из выпуклых вершин многоугольника, т.е. листьев G_M и G_{L^k}. Рассмотрим стянутый граф G_M (граф, в котором вершины каждого ε-кластера стянуты в одну), и мысленно удалим из него все листья. Так как исходный граф был деревом, полученный граф также является деревом, а в любом дереве существует висящая вершина. Т.е. существует ε-кластер C, у которого все инцидентные ребра, кроме одного, ведут в листья. Эта ровно та ситуация, которая проверяется в строчке 20 функции *Перейти*. Следовательно, одна из вершин C будет добавлена в очередь, и обход продолжится. Заметим также, что в этот момент все вершины кластера C уже обработаны.

И так далее, на каждом шаге будет находиться ε-кластер, все кроме одного ребра которого посещены, и одна из его вершин будет добавляться в очередь. Поэтому закончится обход только тогда, когда будет обработан последний кластер.

Для окончания доказательства остается воспользоваться только следствием 6.0.25. □

Лемма 6.0.27. *Если функция* Перейти *вызывается от параметров (u, v, u', v'), то (u, v) - ε-длинное ребро.*

Доказательство. Вызов *Перейти*(u, v, u', v') происходит только тогда, когда ε-кластер $C(u)$ полностью обработан, и все его внутренние ε-короткие ребра посещены. Следовательно, (u, v) - ε-длинное ребро. □

Наконец, мы готовы доказать корректность алгоритма.

Сначала докажем, что ответ "нет" корректен. В процессе алгоритма мы строим итоговое отображение f, заданное на ε-кластерах G_M. Покажем, что оно строится однозначным образом. Иными словами, пусть линейная и срединная оси сильно эквивалентны. Существует биекция F, удовлетворяющая условиям определения 4.0.11. Докажем, что для любого ε-кластера C: $f(C) = F(C)$.

Будем действовать по индукции по ε-кластерам в порядке их обработки. База индукции: для выпуклых вершин p: $f(C(p)) = \{p\} = F(C(p))$.

Далее, рассмотрим переход по ребрам $(u, v) \in E_M$ и $(u', v') \in E_{L^k}$. По индукционному предположению имеем $f(C(u)) = F(C(u))$. Кроме того, аналогичное равенство справедливо для всех кластеров C_i, инцидентных $C(u)$ по помеченным ребрам: $f(C_i) = F(C_i)$, для $i = 1, ..., |C(u)| + 1$. Как мы доказали ранее, переходы будут осуществляться только по ε-длинным ребрам (u, v), т.е. (u, v) - последнее обрабатываемое ε-длинное ребро, идущее из $C(u)$. А так как $u' \in F(C(u))$, по определению сильной эквивалентности отсюда следует, что вершина v' обязана лежать в $F(C(v))$.

Если вершина v' была посещена, в $f(C(v))$ может быть добавлен результат функции $w' =$ *Расширить*(v'). Покажем, что w' также лежит в $F(C(v))$. В момент вызова функции *Расширить* вершине v' инцидентно ровно два посещенных ребра. Пусть (v', w'_1) - оставшееся непосещенное ребро. Тогда вершина w'_1 должна лежать в том же двойственном кластере, что и v', иначе он будет несвязным. Абсолютно аналогично можно рассмотреть и остальные промежуточные вершины на пути от v' к w'. Тем самым и w' лежит в том же двойственном кластере.

Таким образом, в $f(C(v))$ добавляются только те вершины, которые обязаны лежать в $F(C(v))$. Поскольку в конце размеры этих множеств совпадают, имеем $f(C(v)) = F(C(v))$. Индукция доказана.

Далее, докажем, что ответ "да" корректен. Действительно, в процессе обхода мы пройдем по всем ε-длинным ребрам G_M. Пусть (u, v) одно из них, и обработано оно в вызове *Перейти*(u, v, u', v'). В том случае, если v' уже посещена, то есть для нее известен двойственный кластер, в котором она лежит, мы проверяем, не противоречит ли это значению f (строка 4). Иначе мы либо создаем двойственный кластер v', либо добавляем v' в уже существующий. В обоих случаях накладываемые определением 4.0.11 условия на f выполнены. Следовательно, ответ "да" корректен.

Анализ времени работы и используемой памяти. Во-первых, мы можем реализовать структуры данных, таким образом, что совершаемые над ними операции будут выполняться за время $O(1)$.

- Очередь Q реализуем через массив фиксированного размера. Тогда операции добавления в конец, взятия первого элемента и удаления первого элемента будут выполняться за константное время.
- Класс Cluster, содержащий множество вершин графа, можно реализовать, используя массив boolean. Тогда, если занумеровать все вершины обоих графов, добавление элемента в множество и проверка его принадлежности будет выполняться за $O(1)$.
- Отображение f реализуем как массив, содержащий ссылки на двойственные кластеры в G_{L^k}. Для этого мы занумеруем все ε-кластеры и все двойственные кластеры.

Далее, рассмотрим все фазы алгоритма.

1. Обработка очереди Q. Каждая вершина попадает в очередь не более одного раза. Обработка каждой четверки из очереди занимает константное время (не учитывая затраты на функцию *Расширить*), поэтому обход графов занимает линейное время.
2. Вызовы функции *Расширить*. Функция *Расширить* является рекурсивной, но при каждом ее вызове одно из ребер E_M либо E_{L^k} становится посещено, поэтому общее число вызовов линейно. Учитывая, что нерекурсивная часть *Расширить* выполняется за константу, имеем, что и общее время работы всех вызовов линейно.

Таким образом, весь алгоритм работает за время $O(n)$, n - число вершин в многоугольнике. То, что алгоритм требует $O(n)$ памяти следует из того, что общее число вершин и ребер в обоих графах линейно.

Теорема доказана.

□

Замечание 6.0.28. Приведенный алгоритм, вместе с некоторыми дополнениями, можно использовать и для проверки геометрической эквивалентности. Действительно, для данных графов срединной и линейной осей запустим алгоритм проверки сильной эквивалентности. Ответ "нет" автоматически влечет отсутствие геометрической эквивалентности, как мы уже отмечали в пятой части.

Предположим, что сильная эквивалентность имеет место. Это означает, что нам известны все двойственные кластеры, соответствующие ε-кластерам графа срединной оси. Для достижения геометрической эквивалентности должно быть использовано то же отображение f (с точностью до перестановок внутри кластера). Зная эту информацию, постараемся проверить, существует ли такой набор барьеров, обеспечивающий геометрическую эквивалентность.

Для каждого ε-длинного ребра G_M заведем *допустимый интервал*, определяющийся парой точек на ребре. Допустимый интервал имеет смысл множества точек ребра, в которых допустимо

иметь центр барьера. Изначально допустимый интервал положим равным всему ребру, в последствии мы будем добавлять на него дополнительные ограничения.

Рассмотрим произвольный ε-кластер C и двойственный кластер $f(C)$. Далее, для каждого ε-длинного ребра e_1, инцидентного одной из вершин в C, и для каждого ребра линейной оси $e_2 = (u, v)$, оба конца которого лежат в $f(C)$, обновим допустимый интервал e_1. Для этого рассмотрим точки пересечения пяти отрезков – S_1u, S_1v, S_2u, S_2v и uv, – с e_1, где S_1 и S_2 – сайты, порождающие uv. Под отрезком S_iv мы будем понимать отрезок, соединяющий v с ближайшей точкой S_i.

Тем самым мы определим 5, либо меньше, предельных положений барьера на ребре e_1. Некоторые из них могут быть вырожденные: такие, при которых ни один двух отрезков, составляющих барьер, не будет пересекать e_2. Оставшиеся будут добавлять дополнительные ограничения на центр барьера на ребре e_1, то есть на допустимый интервал.

Оси будут геометрически эквивалентными, если после обработки всех кластеров длина допустимого интервала каждого ребра будет больше 0. В противном случае, ответ "нет".

Оценка времени работы алгоритма: рассмотрим произвольный кластер C, пусть $|C| = l$. Нам известно, что число ε-длинных ребер, инцидентных C, равно $l + 2 = O(l)$. Число внутренних ребер графа линейной оси в $f(C)$ также пропорционально l, однако каждое из них может содержать порядка $max\ k_j$ ребер оси (k - вектор скрытых ребер). Поэтому необходимо обработать $O(l^2 \cdot max\ k_j)$ пар ребер, а обработка каждой пары выполняется за $O(1)$. Имеем, таким образом, что общее время работы $T = O(n^2 \cdot max\ k_j)$.

Поскольку на практике число скрытых ребер при каждой вершине невелико (часто не больше 3) [3], время работы можно считать квадратичным. Помимо этого, размер каждого из ε-кластеров часто также оказывается ограничен константой. В этом случае алгоритм проверки геометрической эквивалентности линеен и не уступает по эффективности алгоритму проверки сильной эквивалентности. Этот результат можно оформить в виде теоремы:

Теорема 6.0.29. *Пусть P - простой многоугольник, $\varepsilon > 0$, k - набор скрытых ребер, $max\ k_j$ не превосходит константы. Тогда проверка геометрической эквивалентности $M(P)$ и $L^k(P)$ может быть выполнена за время $O(n^2)$.*

Если при этом размер каждого ε-кластера ограничен константой, проверка может быть выполнена за время $O(n)$.

7 Практическая реализация

Алгоритм проверки сильной эквивалентности был реализован на языке Java (JDK 1.6). Приложение состоит из следующих компонентов:

1. модель включает набор взаимосвязанных классов:
   (a) Vertex - хранит список инцидентных ребер и ссылку на содержащий эту вершину кластер,
   (b) Edge - хранит ссылки на инцидентные вершины,
   (c) Cluster - хранит список вершин кластера,
   (d) Graph - хранит список вершин, ребер и кластеров в графе,
   (e) Input содержит ссылки на два графа - срединной и линейной оси.
2. компонента ввода-вывода состоит из класса InputReader, который считывает данные из входного потока в определенном формате, и возвращает экземпляр Input.
3. центральная компонента - класс Algorithm, реализующий алгоритм, описанный в Теореме 6.0.18.

Для тестирования приложения было подготовлено 11 тестовых случаев.

Также была разработана программа для визуализации алгоритма. Перечислим ее основные конмпоненты.

1. Для интеграции с алгоритмом реализованы классы-наследники ProxyVertex и ProxyEdge, которые вместе с базовыми операциям Vertex и Edge выполняют регистрирование соответствующих событий; также добавлен класс-фабрика GraphFactory для того, чтобы создавать объекты proxy-классов тогда, когда включен режим визуализации.
2. Добавлены классы для каждой сущности, рисуемой на экране: DrawableVertex (вершина), StraightEdge (прямолинейное ребро), ParabolicEdge (параболическое ребро) и др.
3. Классы для хранения и обработки событий: Event, EventBuffer, EventDispatcher.
4. GUI форма VisualFrame для обработки пользовательских событий (реагирование на нажатие клавиши клавиатуры, нажатие на кнопки и др.) и отрисовки осей.

Всего оба приложения включают 25 классов, общий объем кода: 2500 строк.

8 Заключение

В результате данной работы было проведено исследование возможных подходов для оценки сходства срединной оси многоугольника и линейной оси. Было введено понятие сильной эквивалентности, для которого сохраняют силу доказанные ранее теоремы о достаточном условии эквивалентности, и применимы алгоритмы нахождения набора скрытых ребер, на которых достигается эквивалентность. Для нового определения был разработан алгоритм проверки эквивалентности, применимый для всех простых многоугольников, и работающий за линейное время и использующий линейную память. Также был рассмотрен подход, основанный на геометрической близости вершин осей. Было доказано, что данный тип эквивалентности является более сильным, чем два других, и что для него также возможна эффективная проверка для данных срединной и линейной осей.

В качестве дальнейшего направления для исследования можно предложить задачу о нахождении оптимального набора скрытых ребер k, достаточного для сильной эквивалентности. Логичный подход к решению этой задачи заключается в итеративном приближении к ответу. Нам известна оценка сверху и снизу, поэтому применимы методы подбора вектора k, проверки ε-эквивалентности на нем и, в зависимости от результата, сдвиг верхней или нижней границы.

Список литературы

[1] Blum, H.: A Transformation for Extracting New Descriptors of Shape, MIT Press, pp. 362-380, 1967.

[2] Tanase, M., Veltkamp, R.C.: Straight Skeleton Approximating the Medial Axis, Proc. 12th European Symposium on Algorithms, pp. 809–821, 2004

[3] Tanase, M.: Shape Decomposition and Retrieval. Ph.D. Thesis, Utrecht University, 2005.

[4] Трофимов, В.: Построение линейной оси произвольного многоугольника, дипломная работа, СПбГУ, 2006.

[5] Trofimov, V., Vyatkina, K.: Linear Axis for General Polygons: Properties and Computation, Proc. ICCSA'2007, LNCS 4705, Springer-Verlag, pp. 122-135, 2007.

[6] Warnow T.J.: Tree Compatibility and Inferring Evolutionary History, Journal of Algorithms 16:3, Elsevier, pp. 388-407, 1994.

[7] Чурин, А.: Программа "Морфинг полигонов", 2007.