Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Кафедра компьютерных технологий

А. П. Ковшаров

Оптимизация параметров стратегий поиска объектов на море

Бакалаврская работа

Научный руководитель: А. С. Ковалев

Санкт-Петербург
2015

Содержание

Содержание **2**

Введение **4**

Глава 1. Решение массовой задачи о ближайшем отрезке с помощью диаграммы Вороного **6**

1.1 Диаграмма Вороного для точек 6
1.1.1 Определение диаграммы Вороного для точек 6
1.1.2 Построение диаграммы Вороного для точек 7
1.2 Диаграмма Вороного для отрезков 8
1.3 Решение задачи dist2segments 9
1.4 Анализ полученных результатов 9

Глава 2. Решение массовой задачи о ближайшем отрезке с использованием квадродерева **11**

2.1 Квадродерево 11
2.2 Нижняя огибающая 14
2.3 Алгоритм 15
2.3.1 Идея алгоритма 15
2.3.2 Работа алгоритма 15
2.3.3 Оценка числа перебираемых отрезков 18
2.3.4 Анализ полученных результатов 21

Глава 3. Сравнение с другими точными решениями **22**

3.1 Модификация n-grid для точного решения 22
3.2 Сравнительные испытания 22

Глава 4. Применение **25**

4.1 Решение задачи обратного геокодирования 25
4.2 Решение задач интерполяции 26

Заключение **28**

Введение

Задача определения маршрута поиска объекта на первый взгляд очень похожа на задачу коммивояжера. Однако в реальности все подругому и проявляется несколько других особенностей задачи:

- Зачастую все вершины посетить физически невозможно. Соответственно выделяются вершины в которых более вероятно обнаружить объект. Сопоставим вершине v величину p_v — вероятность обнаружить объект в этой вершине. $p_{path} = \sum_{v \in path} p_v$.

Существующие подходы строят весь маршрут исходя из статичных данных в начальный момент времени — информации о начальном распределении и модели его распределения. Таким образом главным недостатком существующих подходов является необходимость разработки нового алгоритма для всех различных моделей распределения. Учитывая тот факт, что подобрать правильную модель, хорошо приближающую реальность, крайне непросто, возникает необходимость разработки алгоритма, работающего единообразно на широком классе различных моделей. Основная идея рассматриваемого подхода — использование симулятора для получения информации о распределении в любой момент времени при построении маршрута. Таким образом при планирование пользователь в первую очереь выберет модель, как можно лучше приближающую реальность в данном случае, а после запустит единственный алгоритм.

Концепт данной задачи был продемонстрирован на одной из выставок. К задаче был проявлен интерес и было решено внедрить ее в комплекс расчетных морских задач.

В главе 1 решаемая задача будет рассмотрена более подробно. Опи-

саны классы маршрутов, получаемые при использовании стратегии “Параллельное галсирование”. ПБудут приведены особенности задачи, которые отличают ее от классической задачи коммивояжера и делают невозможным использование ранее разработанных методов решения TSP для решения исходной задачи в общем случае.

В главе 2 будут рассмотрены вопросы, связанные с разработкой симулятора на CUDA. Обозначены предоставляемые им сервисы.

В главе 3 будет описан алгоритм построения маршрутов согласно стратегии “Параллельное галсирование”.

Глава 1. Решение массовой задачи о ближайшем отрезке с помощью диаграммы Вороного

В данной главе дается определение диаграммы Вороного и предлагается решение задачи dist2segments, использующее диаграмму Вороного для отрезков. Реализацию данного решения можно найти в сборнике библиотек CGAL [?].

1.1. Диаграмма Вороного для точек

Для лучшего понимания диаграммы Вороного для отрезков рассмотрим диаграмму Вороного для точек. Вообще, диаграмма Вороного для точек является частным случаем диаграммы Вороного для отрезков, однако она характерна тем, что может быть представлена РСДС, так как она имеет линейную структуру.

1.1.1. Определение диаграммы Вороного для точек

Сначала дадим определение локусам (locus). Итак, локусом называется область точек, обладающая требуемым свойством [?]. Собственно, диаграмма Вороного для произвольного множества точек – это представление построенной системы локусов для каждой точки p данного множества S со следующим свойством: расстояние от любой точки локуса q до p меньше, чем до любой точки из S (рис. 1.1). Локусы, обладающие данным свойством назовем ячейками диаграммы Вороного. Диаграмма Вороного связана с триангуляцией Делоне [?, ?] следующим образом: граф Делоне является триангуляцией Делоне в предположении, что никакие четыре точки из S не лежат на одной окружности.

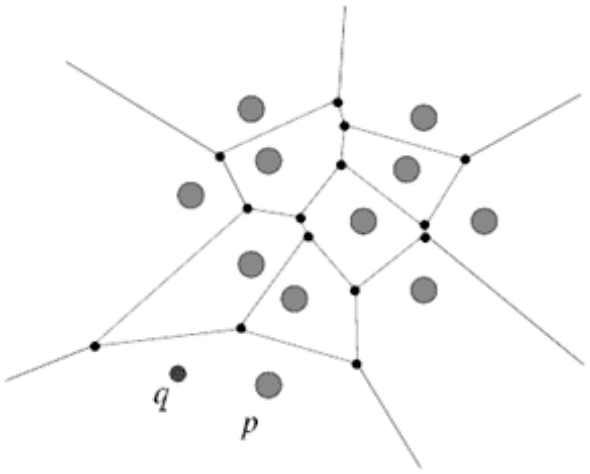

Рис. 1.1: Диаграмма Вороного

1.1.2. Построение диаграммы Вороного для точек

Существует много способов построения диаграммы Вороного. Можно доказать, что нижняя оценка сложности построения диаграммы Вороного есть $O(n \log n)$. Эта оценка достигается, например, через бинарную декомпозицию и последующую композицию задачи (Divide and Conquer) [?]. Собственно, алгоритм построения диаграммы Вороного в данной работе рассмотрен не будет, его можно найти в [?, ?], однако мы рассмотрим подробнее результат работы алгоритма построения диаграммы Вороного. Итак, на выходе обычно получают или саму диаграмму, или двойственную ей триангуляцию Делоне (Delaunay triangulation). Возникает естественный вопрос о представлении результата. Обычно диаграмма Вороного, если ее необходимо представить явно, представляется в виде РСДС [?]. Отметим, что локализация точки в РСДС оценивается в $O(\log n)$, однако, если запросы хорошо кэшируемы, то локализацию можно проводить и за константное время.

Часто применяют метод локальной модели вычислений. Именно, если последовательные запросы на решение задачи dist2segments расположены недалеко друг от друга, то имеет смысл выбрать в качестве стартовой

грани локализации грань, которая была найдена для последнего запроса. Очень часто такая техника кэширования позволяет проводить локализацию в среднем за $O(1)$, так как искомая точка оказывается расположенной или на той же грани, что и предыдущая, или на одной из соседних граней.

1.2. Диаграмма Вороного для отрезков

Для множественного запроса типа dist2segments можно использовать диаграмму Вороного для отрезков (segment Voronoi diagram, SVD) [**?**] (рис. 1.2).

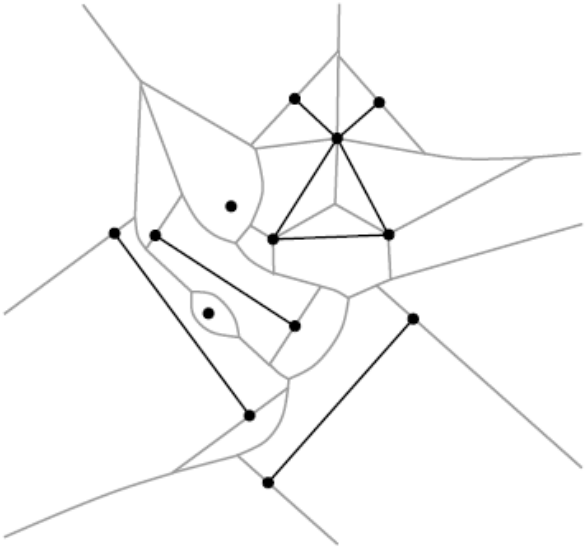

Рис. 1.2: Диаграмма Вороного для отрезков

Диаграмма Вороного для отрезков определяется так же, как и диаграмма Вороного для точек, однако ячейки определяются уже для отрезков. Именно, ячейкой Вороного (гранью) отрезка s называется множество всех точек плоскости, более близких к s, чем к какому-либо другому отрезку s из множества S. На самом деле отличие от диаграммы Вороного для точек очень существенно – границы граней (бисекторы) являются кривыми второго порядка. Как следствие – практически невозможно хранить диаграмму Вороного для отрезков в явном виде (в отличие от диаграм-

мы Вороного для точек, для которой существует представление в виде, например, РСДС). Однако хранить ее в явном виде зачастую и не требуется: очень часто хранят триангуляцию Делоне (constrained Delaunay triangulation, CDT) [?], двойственную диаграмме Вороного. Это важный момент – по триангуляции Делоне диаграмма Вороного восстанавливается однозначно, хотя это и сопряжено с большими трудозатратами.

Наиболее часто используемая реализация диаграммы Вороного принадлежит Geometry Factory – это библиотека коллекции CGAL [?] (segment Voronoi diagram library), которая, в свою очередь, признана классическим трудом в области вычислительной геометрии. Сложность ее построения – $O(n \log n)$ [?] (хотя в CGAL используется инкрементальный алгоритм сложности $O(n \log^2 n)$). Локализация точки в диаграмме Вороного – $O(\log n)$. Однако, применяя технику локальной модели вычислений, можно добиться оценки сложности запроса $O(1)$. В библиотеке CGAL данная техника с успехом была реализована, что позволило существенно сократить время обработки запроса dist2segments.

1.3. Решение задачи dist2segments

На этапе предобработки строится диаграмма Вороного для точек и отрезков. Сложность в худшем случае $O(n \log n)$. Затем для каждой точки-запроса из Q проводим локализацию в построенной диаграмме Вороного. Сложность в худшем случае $O(\log n)$, в среднем – $O(1)$. Получив ячейку Вороного, можно элементарными вычислениями получить расстояние до соответствующего отрезка. Сложность в худшем и среднем случае – $O(1)$.

1.4. Анализ полученных результатов

Итак, первый вариант решения задачи dist2segments требует в худшем и среднем случаях $O(n)$ памяти. Сложность построения диаграммы

Вороного для отрезков в худшем случае – $O(n \log n)$. Локализация точки в ней в худшем случае – $O(\log n)$, возможно решение с кэшированием, которое существенно улучшает результат – сложность в среднем $O(1)$. Собственно, как вариант решения поставленной задачи он является классическим и общепризнанным, но практика показывает, что на многих прикладных задачах лучше пользоваться другими методиками, которые, по сути, изоморфны диаграмме Вороного для отрезков, однако позволяют сократить время запроса на порядок.

Глава 2. Решение массовой задачи о ближайшем отрезке с использованием квадродерева

2.1. Квадродерево

Квадродерево – поисковая структура данных, которая хранит в себе подразбиение плоскости и позволяет быстро производить локализацию точек-запросов. Узел (ячейка) квадродерева представляет собой прямоугольник, для которого определена некоторая мера его насыщенности $p(C)$. Если узел насыщен ($p(C) > T$, где T – предельное насыщение), то происходит его разбиение на четыре одинаковых дочерних узла (делением пополам по вертикали и по горизонтали). Таким образом, в каждый момент времени у узла или нет детей, или их четыре. Разбиение происходит до тех пор, пока все узлы не перестанут быть насыщенными, или не будет достигнута максимальная глубина подразбиения. Ограничение глубины подразбиения играет важную роль в виду того, что не всегда получается сделать узел ненасыщенным за конечное (или разумное) количество разбиений, далее будет дано более точное обоснование необходимости ограничения.

Квадродеревья и их модификации очень часто применяют для решения задач примерного поиска ближайшего соседа (Approximate Nearest Neighbor Search) для точек [**?**] (рис. 2.1). В качестве меры насыщения в этой задаче часто выбирают количество точек, среди которых производится поиск, попавших в ячейку. Насыщенность обычно ограничивают одной точкой в одной ячейке. Максимальная глубина древа для n точек может составлять n, в результате чего время локализации может составлять $O(n)$. Для борьбы с этим была разработана структура данных Skip-Quadtree [**?**], которая позволяет производить локализацию за $O(\log n)$.

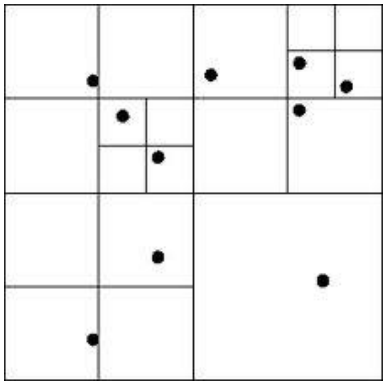

Рис. 2.1: Квадродерево

Квадродерево обладает свойствами, которые делают его предпочтительным для решения задачи dist2segments, по сравнению с другими деревьями. Например, квадродерево не требует наличия логики, по которой будет происходить разбиение ячейки (в отличии от kd-дерева). Разбиение всегда происходит на четыре равные ячейки. Также важным свойством является быстрота локализации точки в нем. За счет того, что квадродерево имеет регулярную структуру, можно по координатам точки и глубине сразу получить путь в квадродереве. Более того, в виду замечательного совпадения, можно за несколько операций над числами с плавающей точкой по координате точки получить путь до листа. Остановимся на этом подробнее.

Рассмотрим устройство чисел с плавающей точкой. Они состоят из знака s, мантиссы m и экспоненты e, причем $m \in [1, 2)$ и первая единица не хранится. Само число представляется в виде $sm2^e$. В компьютерном представлении экспонента хранится в виде двоичной последовательности $b_1 b_2 \dots b_n$, при этом $m = 1 + \sum_{i=1}^{n} b_i 2^{-i}$.

Теперь вернемся к квадродереву. Пусть (x_0, y_0) – координаты левого нижнего угла первого уровня квадродерева, а w, h – его ширина и высота. Перейдем в новую систему координат.

$$\begin{cases} x' = (x - x_0)/w \\ y' = (y - y_0)/h \end{cases}$$

В этой системе координат внутренние точки квадродерева имеют

координаты $(x', y') \in [0,1] \times [0,1]$. Посмотрим, что происходит при переходе на уровень вниз. В системе координат ячейки ее дети имеют координаты $[0, \frac{1}{2}] \times [0, \frac{1}{2}]$ – левый нижний, $[\frac{1}{2}, 1] \times [0, \frac{1}{2}]$ – правый нижний, $[0, \frac{1}{2}] \times [\frac{1}{2}, 1]$ – левый верхний, $[\frac{1}{2}, 1] \times [\frac{1}{2}, 1]$ – правый верхний. Опишем переход в систему координат ребенка.

$$\begin{cases} x_{i+1} = 2(x_i - x_{c_i}) \\ y_{i+1} = 2(y_i - y_{c_i}) \end{cases}$$

Заменим координаты левых нижних углов детей $(x_{c_i}, y_{c_i}) \in \{0, \frac{1}{2}\} \times \{0, \frac{1}{2}\}$ на $(b^x_{c_i}, b^y_{c_i}) = (2x_{c_i}, 2y_{c_i}) \in \{0,1\} \times \{0,1\}$.

$$\begin{cases} x_{i+1} = 2x_i - b^x_{c_i} \\ y_{i+1} = 2y_i - b^y_{c_i} \end{cases}$$

Теперь посмотрим на обратный переход.

$$\begin{cases} x_i = \frac{1}{2}(b^x_{c_i} + x_{i+1}) \\ y_i = \frac{1}{2}(b^y_{c_i} + y_{i+1}) \end{cases}$$

Найдем формулу перехода от уровня d к самому верхнему уровню.

$$\begin{cases} x_1 = \sum_{i=1}^{d} b^x_{c_i} 2^{-i} + x_d \\ y_1 = \sum_{i=1}^{d} b^y_{c_i} 2^{-i} + y_d \end{cases}$$

Что по сути является двоичным представлением координат точки в системе координат верхней ячейки квадродерева. То есть, совершив преобразование координат, мы получаем путь в квадродереве в виде битов двоичного представления координат.

Эта техника позволяет производить быструю локализацию в квадродереве. Специальной обработки требует только экспонента. Также неоценимым достоинством этого метода является его робастность. Если аккуратно произвести преобразование координат мы можем получить путь на глубину $d = 53$ для чисел с плавающей точкой двойной точности.

2.2. Нижняя огибающая

Неформально нижняя огибающая (lower envelope) множества объектов на плоскости – множество точек этих объектов, видимых наблюдателем, расположенным в точке $(0, -\infty)$. Формально же это граф, представляющий из себя поточечный минимум кусочно-заданных функций [?] (рис. 2.2). Также наряду с нижней огибающей часто рассматривается минимизационная диаграмма (minimization diagram), которая представляет собой проекцию нижней огибающей на горизонтальную ось (рис. 2.3).

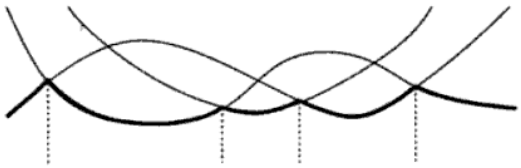

Рис. 2.2: Нижняя огибающая

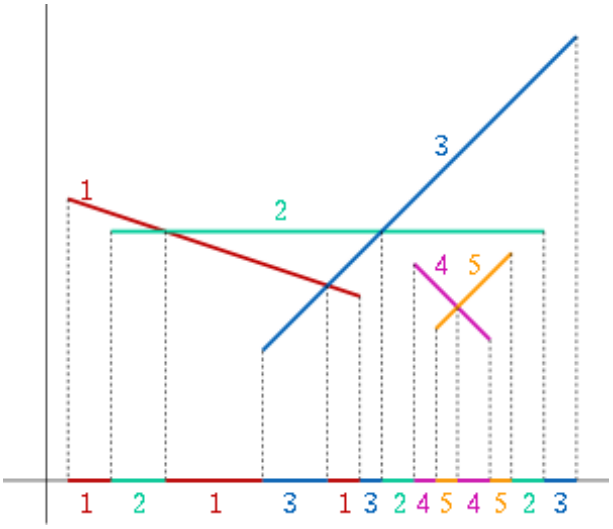

Рис. 2.3: Минимизационная диаграмма

2.3. Алгоритм

2.3.1. Идея алгоритма

Основной идеей всех алгоритмов поиска ближайших сайтов (sites), основанных на подразбиении пространства, является растеризация (с явным построением или без него) диаграммы Вороного в этом подразбиении. После этого в ячейках подразбиения оказывается информация, обо всех ближайших сайтах для всех точек этой ячейки. Поиск ближайшего сайта происходит путем локализации в этом подразбиении, и последующим перебором всех сайтов, ближайших к найденной ячейке.

Ввиду нетривиальности задачи поиска всех сайтов ближайших к ячейке, во многих алгоритмах переходят к неточному решению задачи поиска ближайшего отрезка [?], производя поиск сайтов, ближайших к каким-то точкам ячейки. Точки обычно выбираются таким образом, чтобы обеспечить заданную точность, но в некоторых случаях даже не идет речи о точности [?]. Для некоторых случаев погрешность допустима, но робастность (robustness) является важной характеристикой алгоритмов вычислительной геометрии [?]. Предложенный алгоритм позволяет произвести точный поиск ближайших отрезков для ячеек, при условии, что можно явно (хотя бы кусочно) задать расстояние от границ ячеек до отрезков в виде полинома.

2.3.2. Работа алгоритма

Для отрезков строится ограничивающий прямоугольник (bounding box), этот прямоугольник будет первым уровнем квадродерева. В качестве меры насыщенности узла берется количество ближайших отрезков к данной ячейке. Для первого узла ближайшими будут все отрезки, так как они все лежат внутри. Далее происходит рекурсивное подразбиение узлов. Ближайшие к дочернему узлу отрезки будут среди ближайших к его роди-

телю, так как дочерний узел геометрически лежит внутри родительского. Необходимо произвести фильтрацию лишних отрезков.

Утверждение 2.1. *Ближайшие отрезки для точек ячейки – это отрезки ближайшие к ее границе, и отрезки пересекающие ячейку*

Доказательство. Обозначим: S – множество отрезков, ближайших к ячейке, S_b – ближайших к границе, S_i – пересекающих ячейку.

- $S_b \cup S_i \subset S$

  $S_b \subset S$ – очевидно, так как граница ячейки – это ее подмножество.

  Для любой точки на пересечении отрезка и ячейки этот отрезок будет ближайшим, значит $S_i \subset S$

- $S_b \cup S_i \supset S$

  Предположим, что это не так.

  Пусть s – отрезок, не пересекающий ячейку, и он не является ближайшим ни к одной точке на границе. Пусть он ближайший для точки P ячейки, а Q – точка s, ближайшая к P. Построим отрезок PQ, так как точка P вне ячейки, а точка Q внутри, то PQ пересечет границу ячейки, допустим в точке F. Рассмотрим отрезок s', ближайший к F. Пусть точка E – ближайшая точка на нем к F. Так как s' ближайший к F, то $|FE| < |FQ|$, по неравенству треугольника $|PF| + |FE| < |PE|$. Подставив первое неравенство во второе, мы получим, что отрезок s' ближе к P чем s (рис. 2.4).

  Противоречие.

□

Это простое утверждение показывает, что для фильтрации нам необходимо взять из отрезков только те, которые являются ближайшими для границы ячейки, и те, которые ее пересекают. Для поиска отрезков, ближайших к границе ячейки, для каждой стороны прямоугольника строится нижняя огибающая функций кратчайшего расстояния от стороны до от-

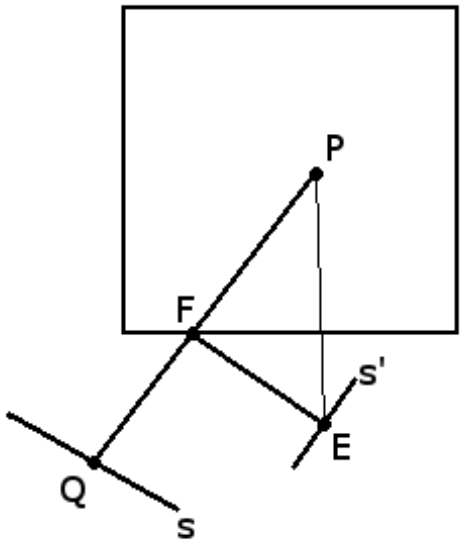

Рис. 2.4: Противоречие

резков, которые фильтруются (рис. 2.5).

Функция кратчайшего расстояния до отрезка состоит из трех частей: двух функции расстояния от стороны до концов отрезка, и функции расстояния от стороны до прямой, содержащей этот отрезок, заданной на ограниченном промежутке. В результате из нижней огибающей можно выделить информацию об отрезках ближайших к сторонам ячейки. Также эта фильтрация оставляет отрезки, пересекающие границу. Значит, к полученным отрезкам остается только добавить отрезки лежащие внутри ячейки. Для проверки этого условия достаточно проверить принадлежность одной из точек отрезка ячейке.

Подразбиение будет происходить до тех пор, пока все ячейки не перестанут быть насыщенными, или пока не будет достигнута максимальная глубина подразбиения. В данной задаче очень важно ограничить глубину подразбиения, так как в вырожденных случаях (degenerate cases) некоторые ячейки подразбить не получится. Вырожденным случаем для диаграммы Вороного является наличие четырех и более сайтов равноудаленных от одной точки. В таком случае в этой точке получается вершина диаграммы

Рис. 2.5: Нижняя огибающая функций кратчайших расстояний

Вороного, граничащая с ячейками соответствующих сайтов. В результате наличия большого числа сайтов расположенных таким образом (пусть их n), ячейка квадродерева, содержащая эту точку, будет ближайшей как минимум к n сайтам. Разбив такую ячейку мы все равно получим одну ячейку содержащую эту вершину диаграммы Вороного. Поэтому имеет смысл ограничивать глубину разбиения. Итак, в результате получается квадродерево, в листьях которого лежит информация о ближайших к ним отрезках. Поиск ближайшего отрезка по такой структуре данных осуществляется в два этапа. Сначала происходит локализация точки-запроса в квадродереве. Затем перебираются все отрезки, ближайшие к найденной ячейке, и среди них выбирается ближайший.

2.3.3. Оценка числа перебираемых отрезков

Скорость обработки запроса очень сильно зависит от числа отрезков в ячейке. Рассмотрим равномерное распределение точек-запросов на

прямоугольнике, задаваемом верхним уровнем квадродерева. Пусть X – случайный запрос, $n(X)$ – число перебираемых отрезков, при запросе X, C – прямоугольник, покрываемый верхним уровнем квадродерева, L – множество листьев квадродерева.

$$E\{n(X)\} = \int_C n(X)dp(X) = \sum_{l \in L} n_l p_l \tag{2.1}$$

Так как распределение равномерное, то $p_l = \frac{S_l}{S}$, где S_l – площадь листа l, S – площадь покрываемая квадродеревом. В итоге получаем простую формулу.

$$E\{n(X)\} = \frac{1}{S} \sum_{l \in L} n_l S_l \tag{2.2}$$

Попытаемся оценить эту величину сверху. Пусть глубина дерева ограничена числом d, а c – насыщенность узла квадродерева, после которой он разбивается. Заметим, что листья бывают двух видов: насыщенные и ненасыщенные. Насыщенные листья – это листья, которые находятся на уровне d и все еще содержат больше, чем c отрезков. Обозначим множество ненасыщенных листьев G (good), множество насыщенных листьев B (bad).

$$E\{n(X)\} = \frac{1}{S} \sum_{l \in L} n_l S_l = \frac{1}{S} \sum_{l \in G} n_l S_l + \frac{1}{S} \sum_{l \in B} n_l S_l \tag{2.3}$$

Оценим первую сумму.

$$\frac{1}{S} \sum_{l \in G} n_l S_l \leq \frac{c}{S} \sum_{l \in G} S_l \leq \frac{c}{S} S = c \tag{2.4}$$

Оценим вторую сумму. Так как листья из B находятся на уровне d, то $S_l = \frac{S}{4^d}$. В каждом таком листе не более n отрезков.

$$\frac{1}{S} \sum_{l \in B} n_l S_l \leq \frac{Sn}{4^d S} |B| = \frac{n|B|}{4^d} \tag{2.5}$$

Осталось оценить мощность множества B. Ячейка попадает в множество B, если ее пересекает большое число ячеек диаграммы Вороного исходного множества отрезков. Это возможно в случае попадания туда вершины

диаграммы Вороного большой степени (рис. 2.6) или пересечения большим числом узких локусов (рис. 2.7). Вершин в диаграмме Вороного $O(n)$, локусов тоже $O(n)$.

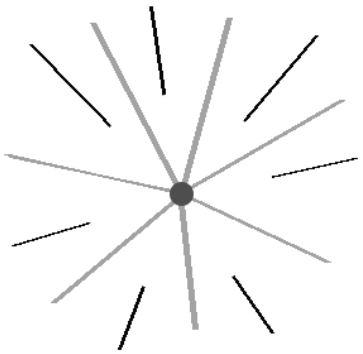

Рис. 2.6: Вершина диаграммы Вороного большой степени

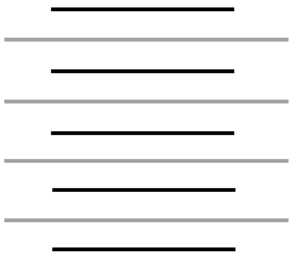

Рис. 2.7: Узкие локусы

Рассмотрим подразбиение прямоугольника, покрываемого верхним уровнем диаграммы Вороного, на 4^d ячеек (2^d по вертикали и горизонтали). Границы ячеек диаграммы Вороного представляют из себя отрезки прямых и парабол, следовательно узкие ячейки растеризуются в этой сетке как отрезки или параболы, а не как площадные объекты, так как они уже ячеек сетки. При растеризации в сетку, параболы и отрезки пересекают $O(n)$ ячеек (сетка $n \times n$). Следовательно узкий локус пересекает $O(2^d)$ ячеек. Можно оценить $|B|$.

$$|B| = O(n2^d) \tag{2.6}$$

Сводя все воедино, получаем верхнюю оценку математического ожидания числа перебираемых отрезков.

$$E\{n(X)\} = c + \frac{nO(n2^d)}{4^d} = c + \frac{O(n^2)}{2^d} \tag{2.7}$$

Отсюда видно, что при $d = 2\log_2 n + C$, где C – константа, $E\{n(X)\} = O(1)$.

2.3.4. Анализ полученных результатов

Если произвести грубую оценку времени построения квадродерева для этой задачи, то получается, что оно ограничено только максимальной глубиной подразбиения. Это так, но на практике построение происходит достаточно быстро, в виду того, что сильное подразбиение испытывают в основном области, содержащие вершины и узкие локусы диаграммы Вороного. Максимально возможное число ячеек будет 4^d, что равно $O(n^4)$.

Время поиска ближайшего отрезка складывается из времени локализации и времени поиска ближайшего отрезка среди ближайших к ячейке. Тогда как первая величина ограничена сверху d, вторая ограничивается только количеством отрезков (достаточно вспомнить вырожденный случай). Ввиду того что d обычно не очень велико и локализация в дереве, как было показано, не требует на каждом шаге операций с плавающей точкой, а происходит простой спуск по заданному пути, основной вклад во время поиска ближайшего вносит перебор отрезков из ячейки. Хоть локализация и занимает $O(\log n)$ времени, но, по уже оговоренным причинам, алгоритм на разумном числе отрезков оказался очень эффективным, в виду того, что математическое ожидание перебираемых отрезков $O(1)$. Практические данные показывают, что эта величина лежит в промежутке $[0.5c, c]$, где c – предельная насыщенность ячейки.

Максимальное количество занимаемой памяти можно грубо оценить, как $O(2^d n + 4^d) = O(n^4)$, что на практике не наблюдается.

Глава 3. Сравнение с другими точными решениями

3.1. Модификация n-grid для точного решения

В работе [?] была предложена структура данных n-grid, позволяющая эффективно решать задачу поиска ближайшего отрезка. Автором этой работы было предложено несколько алгоритмов фильтрации не ближайших отрезков из ячеек сети. Но предложенные алгоритмы или осуществляют очень грубую фильтрацию, оставляя при этом много лишних отрезков и не ухудшая точность поиска ближайшего отрезка, или переходят к неточному решению задачи поиска ближайшего отрезка, отфильтровывая, возможно, ближайшие отрезки. Для этой структуры данных мной была применена та же схема фильтрации, что и в квадродереве. После этого незначительного изменения данная структура данных позволила решать задачу поиска ближайшего отрезка точно и оптимально, в смысле количества отрезков, которое нужно рассматривать перебором

3.2. Сравнительные испытания

В сравнении будут участвовать:

- SVD (реализация из библиотеки CGAL);
- модифицированный n-grid (модифицированная мною, реализация, используемая в компании Транзас);
- квадродерево (моя реализация).

Все испытания проводились на случайно генерируемых данных. Для генерации данных бралась прямоугольная область пространства, в которой из случайной точки этой области начинала генерироваться случайная цепь.

Случайная цепь характеризуется максимальной и минимальной длинами шага, а также максимальным и минимальным углами поворота. При выходе цепи за пределы области начинается генерация новой цепи. В результате получается нечто, напоминающее картографические данные.

Для определения скорости поиска ближайшего отрезка производится один миллион случайных запросов, по которым усредняется время поиска.

В табл. 3.1, табл. 3.2, табл. 3.3, табл. 3.4 приведены усредненные результаты сравнительных испытаний.

Таблица 3.1: Скорость поиска ближайшего отрезка в миллисекундах

Алгоритм/количество отрезков	100	1000	10000	100000
quadtree	0.00152	0.00148	0.00165	0.00268
n-grid	0.00172	0.00165	0.00186	0.00373
SVD	0.04676	0.04495	0.07017	0.08502

Таблица 3.2: Время препроцессирования в секундах

Алгоритм/количество отрезков	100	1000	10000	100000
quadtree	1.646	26.524	482.523	7027.799
n-grid	6.716	86.304	1917.237	25193.771
SVD	0.016	0.226	5.093	106.752

Таблица 3.3: Количество памяти, занимаемой структурой, в мегабайтах

Алгоритм/количество отрезков	100	1000	10000	100000
quadtree	1.154	2.988	19.386	184.775
n-grid	1.150	2.061	9.267	65.876
SVD	0.223	1.065	5.982	53.277

Таблица 3.4: Максимальное количество памяти, потребляемое при построении в мегабайтах

Алгоритм/количество отрезков	100	1000	10000	100000
quadtree	1.249	4.010	28.751	266.834
n-grid	1.659	7.387	174.917	2716.963
SVD	0.223	1.065	5.982	53.277

Из результатов сравнения заметно, что многоуровневая сеть и квадродерево работают на порядок быстрее, чем диаграмма Вороного для отрезков. Время, затрачиваемое на препроцессинг, и потребление памяти больше, но это цена за скорости работы самой структуры. Хоть квадродерево и занимает больше памяти, чем многоуровневая сеть, но оно дает выигрыш в скорости поиска ближайшего отрезка и потребляет гораздо меньше памяти во время построения.

В табл. 3.5 приведены усредненные характеристики квадродеревьев, полученных при испытаниях.

Таблица 3.5: Характеристики квадродеревьев

Характеристика/количество отрезков	100	1000	10000	100000
Количество ячеек	47	646	12963	203109
Среднее число отрезков в ячейке	13.74	12.88	12.64	13.39
Максимальная глубина	4	7.25	11	12

Глава 4. Применение

4.1. Решение задачи обратного геокодирования

Задача обратного геокодирования (reverse geocoding) состоит в определении адреса по координатам точки.

Входными данными к этой задаче обычно является множество объектов с известными географическими адресами. Поиск страны и некоторых других частей адреса осуществляется за счет локализации в подразбиении карты мира на страны, страны на регионы и так далее. Для решения данной задачи известно множество методов. Локализация на уровне улиц происходит иным образом. Определение улицы, рядом с которой расположена точка, осуществляется за счет поиска ближайшей улицы или дома (рис. 4.1).

Эти данные представимы в виде отрезков, поэтому можно предобработать данные карты, поместив геометрию карты в структуру для быстрого поиска ближайших отрезков, и запомнив соответствие между геометрическими данными и их метаинформацией.

В результате, с помощью реализованного алгоритма, можно быстро решать задачу обратного геокодирования.

Рис. 4.1: Ближайший объект

4.2. Решение задач интерполяции

Быстрый поиск ближайших отрезков часто используется при интерполяции функций, заданных на линейных объектах. Например, можно восстанавливать высотную модель по изолиниям.

Также, зная расстояние до ближайшей линии, можно варьировать параметры интерполяции. В работе [?] описано применение кампанией Транзас алгоритма поиска ближайшего отрезка для восстановления рельефа по картографическим данным. Они накладывают на известные данные шумы, зависящие от расстояния до изолиний. Вблизи изолиний используется высокочастотный шум низкой амплитуды, симулирующий мелкие неровности рельефа, тогда как на удалении от них используется низкочастотный шум большей амплитуды, позволяющий получать холмистую местность. Этот подход позволяет по картографическим данным получать реалистичный рельеф.

В этой же работе описано использование информации о ближайших отрезках для простого текстурирования рельефа. В зависимости от расстояния до линейных объектов применяются различные текстуры, которые плавно переходят друг в друга. Например, в зависимости от расстояния

до береговой линии, сначала может идти текстура песка, а потом травяной растительности.

Заключение

Как было показано в данной работе, задача определения кратчайшего расстояния до произвольной конфигурации отрезков на плоскости представляет достаточный практический интерес. Были подробно рассмотрены применяемые на практике подходы. Предложенный подход не только позволяет искать ближайшие отрезки быстрее общепринятых методов, но еще и гораздо проще в реализации. Из недостатков данного подхода стоит отметить лишь немного большее, по сравнению с другими рассмотренными методами, потребление памяти. Но для многих задач данные затраты оказываются вполне приемлемыми.

В данный момент большой практический интерес представляет ускорение построения нижних огибающих за счет отказа от точной арифметики в пользу более быстрых альтернативных техник, таких как, например, adaptive precision arithmetic [**?**]. Это позволит сократить время на построение, как минимум в десять раз. Также интересным было бы разработать технику отсечения заведомо далеких отрезков перед построением нижних огибающих. Что более важно, реализованный алгоритм очень просто поддается распараллеливанию, ввиду полной независимости обработки поддеревьев.

Также хотелось бы отметить, что хоть данный метод применялся для отрезков, он легко обобщается и для других классов объектов. К тому же данный подход применим и для других классов задач, таких как поиск наиболее удаленного отрезка (furthest segment problem) и поиск k ближайших соседей (k-Nearest Neighbor Search). Также данный метод не требует построения пересечений отрезков для своей работы, в отличии от других рассмотренных методов.