Санкт-Петербургский государственный университет

Математико-механический Факультет
Направление "Системное программирование"

Сенюков Леонид Владимирович

Реализация вероятностного алгоритма вычислительной топологии

Курсовая работа

Научный руководитель:
д. ф.-м. н., профессор Терехов А. Н.

Консультант:
Dr. Rer. Nat. Федоров Л. А.

Санкт-Петербург
2020

Оглавление

Введение

Вычислительная топология как наука занимается применением современных алгоритмов и структур данных к вычислительным задачам топологии. Но не меньший интерес представляет использование некоторых топологических понятий в различных наукоемких областях. Одно из первых применений вычислительной топологии возникло из задачи восстановления двумерной поверхности из конечного набора точек, взятых из нее, заданных своими координатами, возникшей как способ описания молекулярной поверхности протеинов по данным о центрах и Вандерваальсовых радиусах атомов, составляющих молекулу [6].

В том числе, для описания метрических данных топологическими методами используются инварианты, наиболее изученным из которых является персистентый модуль [7], который на практике можно рассматривать просто как однозначно-заданное свойство конкретного облака точек с метрикой. Для рассмотрения облака точек как единого топологического объекта, с помощью метрики к нему алгоритмически добавляется информация о связности, ведь тогда получающийся объект можно рассматривать уже как граф, вершинами которого являются точки облака.

Хотя процесс проведения ребер в графе может быть формализован произвольно, целью этого построения является задание симплициального комплекса и следующее за этим изучение его инвариантов, например, как признаков метрических данных. В этом смысле наиболее популярными являются построения комплексов Чеха и Вьеториса-Рипса. Алгоритм их построения из облака точек и метрики содержит в себе информацию о фильтрации комплекса, то есть возрастающей цепочке вложений подкомплексов. Фильтрация, таким образом, содержит в себе в том числе и геометрическую информацию об изначальном облаке точек, которая закодирована в виде вычисляемой на ней так называемой персистентной диаграммы [7].

Вычисление персистентных диаграмм (и двойственных им персистентных ландшафтов) фильтрации представляет собой весьма затрат-

ную по времени задачу [8]. Поэтому предлагается использовать вероятностный метод, при котором из облака точек многократно делается выборка фиксированного размера и вычисления производятся на ней, а затем результаты усредняются [3].

При таком подходе к задаче открывается возможность для ускорения работы алгоритма путем распараллеливания вычислений.

1. Постановка задачи

Целью данной работы является реализация алгоритма, применимого для решения задачи построения персистентного ландшафта вероятностным методом [3].

Для достижения этой цели поставлены следующие задачи:

1. Изучить предметную область
2. Реализовать алгоритм, получающий на вход облако точек и строящий по нему усредненный персистентный ландшафт
3. Разработать набор тестовых входных данных
4. Провести тестирование

2. Обзор предметной области

2.1. Основные понятия

Симплексом σ_{n-1} размерности $n-1, n \in \mathbb{N}$, имеющим своими вершинами n точек из $\mathbb{R}^m$, называется множество

$$\{\lambda_1 x_1 + \ldots + \lambda_n x_n : \sum_k \lambda_k = 1, \lambda_k \geq 0\}$$

Грань σ_i симплекса σ_j – выпуклая оболочка подмножества его вершин, которую также будем обозначать $\sigma_i \subset \sigma_j$.

Симплициальный комплекс K с множеством вершин $\mathbb{X} \subset \mathbb{R}^m$ – такой набор симплексов с вершинами $\mathbb{X}$, что для каждого симплекса $\sigma_j \in K$ все его грани $\sigma_i \subset \sigma_j$ также принадлежат комплексу K, и любое непустое пересечение симплексов является гранью.

Часто на практике данные могут представлять из себя конечное метрическое пространство. То есть вместе с множеством вершин $\mathbb{X} \subset \mathbb{R}^m$ дана функция расстояния $d_{\mathbb{X}}(x_k, x_{k'})$ между каждой парой вершин $\{x_k, x_{k'}\} \subset \mathbb{X}$. Тогда с алгоритмической точки зрения нам достаточно рассматривать симплекс как набор точек фиксированного размера $\sigma_i = [x_0, \ldots, x_n]$, имея в виду что $d_{\mathbb{X}}$ всегда хорошо определена и задана отдельно.

Для рассматриваемого конечного метрического пространства $(\mathbb{X}, d_{\mathbb{X}})$ комплексом **Вьеториса-Рипса** с радиусом α назовем такой симплициальный комплекс K с вершинами $\mathbb{X}$, что если для каждой пары из произвольного набора вершин $[x_0, \ldots, x_n]$ выполняется $d_{\mathbb{X}}(x_k, x_{k'}) \leq \alpha$ (где k и $k^{'}$ индексируют все n вершин), то $\sigma_i = [x_0, \ldots, x_n]$ включается как симплекс комплекса K. Для удобства можно обозначить такой комплекс K как $Rips_\alpha(\mathbb{X})$, подразумевая, что функция расстояния задана. Для краткости, алгебраические подробности в этом изложении опущены (они описаны в обзоре [4]), и следующие понятия определены с помощью матричных вычислений.

Матрицей граничного оператора M_p размерности p для комплекса K называется матрица, описывающая, какие грани размерности

p имеют все симплексы размерности $p+1$:

$$M_p[i][j] = \begin{cases} 1, & \text{если } \sigma_i \subset \sigma_j \text{ и } dim(\sigma_i) = dim(\sigma_j) - 1 = p \\ 0, & \text{иначе} \end{cases}$$

Тогда можно задать **Число Бетти** β_p размерности p симплициального комплекса:

$$\beta_p = n_p - rankM_p - rankM_{p+1},$$

где n_p - число симплексов размерности p.

Практически числа Бетти для данного $Rips_\alpha(\mathbb{X})$ интересны тем, они являются гомотопическими инвариантами комплекса, который в данном случае строится на метрических данных. При этом, так как β_0 равно количеству связных компонент комплекса, этот инвариант можно интерпретировать как количество "кластеров". Соответственно β_1 - количество "петлей", β_2 - количество "пустот" и так далее. То есть, знание всей последовательности чисел Бетти дает более полную информацию о метрических данных, чем лишь количество кластеров. Тонкость заключается в том, что правильное значение α неизвестно.

Фильтрацией называется последовательность вложений симплициальных комплексов [7].

Заметим что для данных $(\mathbb{X}, d_\mathbb{X})$ комплекс $Rips_\alpha(\mathbb{X})$ вложен в комплекс $Rips_{\alpha'}(\mathbb{X})$ если $\alpha < \alpha'$. Оказывается, что если α принимает все значения в пределах $[0, +\inf)$, то при построении комплексов Вьеториса-Рипса числа Бетти меняются лишь конечное число раз – при прохождении "критических" значений параметра α. То есть, на $(\mathbb{X}, d_\mathbb{X})$ строится фильтрация $Rips_{\alpha_0} \subset Rips_{\alpha_1} \subset \cdots \subset Rips_{\alpha_r}$, где r - максимальное значение $d_\mathbb{X}$.

Информация об изменениях чисел Бетти при увеличении параметра α и используется для характеризации данных. Для этого редуцируется матрица граничного оператора, определенного каноническим образом, заданного уже на всей фильтрации, а результат отражается в так называемой персистентной диаграмме.

2.2. Персистентная диаграмма

Персистентная диаграмма – один из способов описания облака точек с топологической точки зрения, отражающий информацию об интервалах постоянства чисел Бетти, возникающих в процессе фильтрации.

Иначе говоря, это набор интервалов (b, d), где границы интервалов соответствуют значениям параметра, при которых значение числа Бетти рождается и умирает.

В случае фильтрации Вьеториса-Рипса в качестве границ интервала принимаются максимальное по длине ребро симплекса и максимальное ребро симплекса более высокой размерности, гранью которого является рассматриваемый.

Стандартный алгоритм построения персистентной диаграммы [8]:

- Построение фильтрации – те получение упорядоченного набора симплексов (симплекс в фильтрации идет после своих граней и симплексы одной размерности упорядочены по увеличению "времени" появления)

- Составление матрицы граничного оператора

- Редуцирование матрицы (вариация алгоритма Гаусса над конечным полем)

  В случае, когда вычисления проводятся над $\mathbb{Z}_2$, стандартный алгоритм редуцирования сводится к произведению операции xor над столбцами матрицы "справо налево".

  Существуют также модификации алгоритма, ускоряющие вычисления за счет знания специальной структуры матрицы, реализованные в PHAT [9].

  Именно редуцирование позволяет найти критические значения параметра фильтрации, при которых менялись числа Бетти.

- Считывание персистентной диаграммы через персистентные пары – те пары индексов (i, j), где j – номер столбца редуцированной матрицы, i – индекс нижней единицы в нем

2.3. Персистентный ландшафт

Каждому интервалу (b, d) персистентной диаграммы соответствует точка на плоскости

$$p = (x, y) = (\frac{b+d}{2}, \frac{d-b}{2})$$

Тогда персистентный ландшафт представляет собой набор ломаных

$$\lambda_D(k, t) = k\underset{p}{max}\Lambda_p(t), t \in [0, T], k \in \mathbb{N}$$

где $kmax$ – k по величине значение в наборе и

$$\Lambda_p(t) = \begin{cases} t - b, & t \in [b, \frac{b+d}{2}] \\ d - t, & t \in (\frac{b+d}{2}, d] \\ 0, & \text{иначе} \end{cases}$$

Находясь во взаимно-однозначном соответствии с персистентными диаграммами, персистентный ландшафт представляет собой более удобный инструмент для работы, так как, в отличие от диаграмм, средний персистентный ландшафт может вычисляться как среднее арифметическое нескольких персистентных ландшафтов (рассматриваемых как наборы ломаных разных уровней на плоскости).

2.4. Вероятностный подход

Возможность применения сэмплирования к задаче вычисления персистентного ландшафта подробно описывается в статье [3].

Алгоритм можно формализовать в виде трех этапов:

- Генерируем выборки из точек облака
- Считаем персистентные диаграммы на точках каждой выборки и преобразуем их в персистентные ландшафты
- Вычисляем средний от полученных персистентных ландшафтов

Таким образом, аппроксимированный ландшафт на облаке из N точек можно посчитать за $\mathcal{O}(n \cdot exp(m))$ времени, сделав n выборок размера m. Для сравнения, вычисление полного ландшафта заняло бы $\mathcal{O}(exp(N)$ [3].

2.5. Обзор существующих инструментов

На данный момент одной из самых часто использующихся библиотек топологического анализа данных является GUDHI [5], в которой, помимо прочих инструментов, собраны алгоритмы построения симплициальных комплексов при помощи различных фильтраций.

Если в качестве фильтрации рассматривается цепочка вложений комплексов Вьеториса-Рипса, то лучшей производительности можно достичь, используя пакет Ripser [10], специализирующийся на построении персистентной диаграммы для этой конкретной фильтрации.

Также следует отметить библиотеку PHAT [9], в которой собраны алгоритмы редуцирования матриц, в том числе с поддержкой OpenMP.

3. Описание решения

3.1. Первый подход

Из соображений нахождения наилучшего баланса между сложностью разработки и скоростью работы программы в качестве языка программирования для разработки был выбран C++.

В качестве первого подхода к решению задачи была выбрана стратегия полной независимости вычислений на различных выборках. Для возможности контролировать время и максимальный одновременный расход памяти программой было принятно решение использовать пул потоков [13]. В рамках данного подхода реализовано две версии вероятностного алгоритма – на основе алгоритмов, имеющихся в GUDHI [5] (считаем его эталонным с точки зрения эффективности), и с использованием Ripser [10]. После получения данных об облаке точек, желаемом размере выборок и их количестве точки нумеруются, случайным образом выбирается их подмножество размера равного величине выборки. Вычисления диаграмм на полученных подоблаках точек далее происходят в отдельных потоках при помощи средств соответствующих пакетов, которые затем при помощи GUDHI [5] преобразуются в ландшафт и на них считается средний.

3.2. Второй подход

В качестве второго подхода была выбрана стратегия вычисления диаграмм, при которой результаты редуцирования матриц переиспользуются при вычислениях на последующих выборках.

3.3. Общее симплициальное дерево

Для хранения информации о фильтрации была разработана и реализована структура данных "Общее симплициальное дерево", которое является разделяемым между потоками ресурсом.

Данная структура данных является вариацией бора – ведь каждый

симплекс можно закодировать в виде строки из возрастающих номеров составляющих его точек.

Тогда информацию, характеризующую симплекс, можно сохранить в вершине дерева, соответствующей ему.

Структура поддерживает следующие операции (помимо обращения к специфической информации о симплексе):

1. **вставка** симплекса размерности n занимает $\mathcal{O}(n)$ времени

2. **восстановление** симплекса по указателю на узел в дереве

   Для этого каждый узел также хранит указатель на предшественника.

   Тогда операция восстановления симплекса занимает линейное время от его размерности.

3. **нахождение симплексов-граней**

   Умея восстанавливать симплекс по указателю на узел, мы можем, поочередно исключая входящие в него точки, найти в дереве все его симплексы-грани.

   Так как граней у симплекса размерности n ровно $n + 1$, то такая операция займет суммарно $\mathcal{O}(n^2)$ времени.

"Общее симплициальное дерево" не поддерживает **удаление** элементов из него.

С целью обеспечения потокобезопасности, указатели на последующие элементы дерева хранятся в *tbb::concurrent_unordered_map* [12].

3.4. Построение фильтрации

Задача построения фильтрации Вьеториса-Рипса сводится к нахождению в графе точек, ребер, треугольников, и так далее – те всех не более чем n-мерных тетраэдров до какого-то фиксированного n.

Если представить каждый n-мерный тетраэдр в виде битовой маски, в которой единицы соответствуют содержащимся в нем вершинам,

то нахождение всех $(n+1)$-мерных тетраэдров, включающих в себя рассматриваемый n-мерный, сводится к выполнению операции *and* над строками матрицы смежности графа, полученного проведением ребер между точками облака, соответствующими вершинам тетраэдра и битовой маски, представляющей тетраэдр.

Для представления битовых масок был использован
класс *boost::dynamic_bitset* из [1].

Рассматривая все тетраэдры графа в порядке увеличения размерности, мы постепенно заполним симплициальное дерево. Тогда фильтрация на каждой выборке есть упорядоченная последовательность указателей на его узлы.

Для того, чтобы получить фильтрацию, необходимо отсортировать указатели по увеличению размерности, а при ее равенстве – по увеличению времени рождения симплекса.

3.5. Получение матрицы граничного оператора

После построения фильтрации мы имеем упорядоченную последовательность указателей, по которым естественным образом строится матрица граничного оператора.

Умея по указателю на узел дерева находить все его грани, мы заполним матрицу граничного оператора и запустим ее редукцию при помощи PHAT [9].

Считав полученные персистентные диаграммы, найдем по ним ландшафты и вычислим их среднее при помощи GUDHI [5].

Информацию о том, какие колонки, соответствующие симплексам высшей размерности, занулились, мы сохраняем в дереве, помечая соответствующие им его узлы.

Тогда такие симплексы можно не включать в фильтрацию при ее построении на последующих выборках, уменьшив тем самым общее число симплексов фильтрации (и размер матрицы).

Дело в том, что пустые столбцы, соответствующие симплексам высших размерностей в редуцированной матрице, не соответствуют ника-

кой персистентной паре [8].

3.6. Тестирование

Для тестирования был подготовлен набор входных данных, включающий в себя облака точек, полученные с поверхности нескольких трехмерных объектов (кролик, человек, тор, сфера) из [5] и набор данных с датчиков движения, установленных на людях, занимавшихся различными видами активности (игра в баскетбол, подпрыгивание и так далее) из [2].

Алгоритмы без сэмплирования, а также полученные реализации были протестированы на облаке точек размера 500, полученного с поверхности тела человека, при радиусе фильтрации r = 0.5. Все координаты точек были предварительно нормированы по максимальной из них (не превышают 1).

Для вероятностной аппроксимации из облака делалось 10 выборок размера 200 точек, которые обрабатывались 4 потоками-исполнителями.

Тестирование проводилось на Macbook Pro с 2.6GHz 4-ядерным процессором 16 ГБ RAM.

Алгоритм	Время работы (мс)	CO (мс)
GUDHI	119 283	1165
Ripser	3 140	151
Алгоритм на основе GUDHI	9751	502
Алгоритм на основе Ripser	667	47
Альтернативный алгоритм	63 712	950

Результаты тестирования подхода переиспользования результатов редуцирования:

- Среднее число симплексов высшей размерности, обрабатываемых в рамках одной выборки – 2 000 000
- Не стали включать в фильтрацию суммарно 1 000 000 симплексов

- Благодаря переиспользованию результатов вычислений на выборках удалось уменьшить размеры редуцируемых матриц на 5%
- Заметного прироста производительности подход не принес

В результате тестирования самой эффективной оказалась вероятностная реализация на основе Ripser.

Заключение

- Изучены основные подходы, применяемые в области
- Разработана и реализована структура данных "Общее симплициальное дерево"
- Реализовано две версии алгоритма, в том числе на основе разработанной структуры данных
- Разработан набор тестов для оценки эффективности решений
- Проведено тестирование и получены данные об эффективности решений [11]

Список литературы

[1] Boost – URL: https://www.boost.org (дата обращения 18.05.2020).

[2] Dataset of activities – URL: http://archive.ics.uci.edu/ml/datasets/Daily+an
(дата обращения 18.05.2020).

[3] F. Chazal, B. Fasy, F. Lecci, B. Michel, A. Rinaldo, L. Wasserman, Subsampling methods for persistent homology // Proceedings of the 32nd International Conference on Machine Learning, PMLR 37:2143-2151. — 2015.

[4] G. Carlsson, Topology and Data. // Bull. Amer. Math. Soc. — 2009.

[5] GUDHI – URL: https://gudhi.inria.fr (дата обращения 18.05.2020).

[6] H. Edelsbrunner, D. Letscher, A. Zomorodian, Topological persistence and simplification // Proceedings of the 41st Annual Symposium on Foundations of Computer Science. — 2002.

[7] H. Edelsbrunner, D. Morozov, Persistent Homology: Theory and Practice. // Proceedings of the European Congress of Mathematics. — 2012.

[8] N. Otter, M. Porter, U. Tillmann, P. Greenrod, H. Harrington, A Roadmap for the Computation of Persistent Homology. // EPJ Data Sci, 2017.

[9] PHAT – URL: https://github.com/blazs/phat (дата обращения 18.05.2020).

[10] Ripser – URL: https://github.com/Ripser/ripser (дата обращения 18.05.2020).

[11] Sampling – URL: https://github.com/leoneed03/SamplingLandscape (дата обращения 18.05.2020).

[12] TBB – URL: https://github.com/oneapi-src/oneTBB (дата обращения 18.05.2020).

[13] Thread pool – URL: https://github.com/inkooboo/thread-pool-cpp (дата обращения 18.05.2020).