**BIRCH: эффективный метод кластеризации данных для очень больших баз данных**

|  |  |  |
| --- | --- | --- |
| Отдел компьютерных наук Тянь Чжан  Унив. Висконсин-Мэдисон zhang@cs.wisc.edu | Отдел компьютерных наук Рагху Рамакришнан.  Univ. of Wisconsin-Madison raghuCQlcs. wisc.edu | Департамент компьютерных наук Мирона Ливны \*  Унив. Висконсин-Мэдисон miron@cs.wisc.edu |

**Резюме**

Поиск полезных закономерностей в больших наборах данных вызвал значительный интерес в последнее время, и одной из наиболее широко изученных проблем в этой области является идентификация кластеров или густонаселенных регионов в многомерном наборе данных. Предшествующая работа недостаточно решает проблему больших наборов данных и минимизации затрат на ввод-вывод.

В этой статье представлен метод кластеризации данных под названием BIRCH (сбалансированное итеративное сокращение и кластеризация с использованием иерархий), и демонстрируется, что он особенно подходит для очень больших баз данных. BIRCH инкрементно и динамически кластеризует входящие многомерные метрические точки данных, чтобы попытаться обеспечить наилучшее качество кластеризации с имеющимися ресурсами (т.е. имеющуюся память и временные ограничения). BIRCH обычно может найти хорошую кластеризацию с помощью одного сканирования данных и дополнительно улучшить качество с помощью нескольких дополнительных сканирований. BIRCH также является первым алгоритмом кластеризации, предложенным в области базы данных для эффективной обработки «шума» (точек данных, которые не являются частью базового шаблона).

Мы оцениваем эффективность BIRCH в отношении времени/пространства, чувствительность к порядку ввода данных и качество кластеризации с помощью нескольких экспериментов. Мы также представляем сравнение производительности BIRCH и CLARA NS, метода кластеризации, предложенного недавно для больших наборов данных, и показываем, что BIRCH неизменно превосходит.

**1 Введение**

В этой статье мы рассмотрим кластеризацию данных, которая является определенной проблемой интеллектуального анализа данных. При большом наборе многомерных точек данных резервные данные обычно заняты неравномерно. Кластеризация данных определяет разреженные и переполненные места и, следовательно, обнаруживает общие шаблоны распределения набора данных. Кроме того, полученные кластеры могут быть визуализированы более эффективно и эффективно, чем исходный набор данных [Lee81, DJ80].

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\* Это исследование было поддержано NSF Grant IRI-9057562 и NASA Grant 144-EC78.

Разрешение на изготовление цифровой/печатной копии части или всей этой работы для личного или классного использования предоставляется без платы при условии, что шифры не сделаны или не распространены для получения прибыли или коммерческой выгоды, появляется уведомление об авторских правах, название публикации и ее дата, а также уведомление о том, что копирование осуществляется с разрешения ACM, Inc. Для копирования в противном случае, повторной публикации, публикации на серверах или распространения в списки требуется предварительное разрешение и/или сбор.

SIGMOD '96 6/96 Монреаль, Канада

© 1996 ACM 0-89791-794-4/96/0006... $3.50

Как правило, в кластеризуемых данных участвуют два типа атрибутов: метрические и неметризированные1. В этой статье мы рассмотрим метрические атрибуты, как в большинстве литературы Статистики, где проблема кластеризации -формализована следующим образом: (например, -взвешенное общее/среднее расстояние между парами точек в кластерах), нам предлагается найти раздел набора данных, который минимизирует значение измерительной функции. Это задача неконвексной дискретной [KR90] оптимизации. Из-за обилия локальных минимумов, как правило, нет возможности найти глобальное минимальное решение, не пробуя все возможные разделы.

Мы принимаем определение проблемы, используемое в статистике, но с дополнительным, ориентированным на базу данных ограничением:

Объем доступной памяти ограничен (как правило, намного меньше, чем размер набора данных), и мы хотим минимизировать время, необходимое для ввода-вывода. Связанный момент заключается в том, что желательно иметь возможность учитывать величину мелодии, которую пользователь готов ждать результатов алгоритма кластеризации.

Мы представляем метод кластеризации под названием BIRCH и демонстрируем, что он особенно подходит для очень больших баз данных. Его стоимость 1/0 линейна по размеру набора данных: одно сканирование набора данных дает хорошую кластеризацию, и один или несколько дополнительных проходов могут (необязательно) использоваться для дальнейшего улучшения качества.

Оценивая время/пространство BIRCH, -чувствительность к порядку ввода данных и качество кластеризации, а также сравнивая с другими существующими алгоритмами с помощью экспериментов, мы утверждаем, что BIRCH является наилучшим доступным методом кластеризации для очень больших баз данных. Архитектура BIRCH также предоставляет возможности для параллелизма, а также для интерактивной или динамической настройки производительности на основе знаний о наборе данных, полученных в ходе выполнения. Наконец, BIRCH является первой кластеризацией изменений параллельно предложил в области базы данных, которая обращается к выбросам (интуитивно, точкам данных, которые следует рассматривать как «шум») и предлагает правдоподобное решение.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1Неформально, метрический атрибут - это атрибут, значения которого удовлетворяют требованиям евклидова пространства, т.е. самоидентичность (для любого ) и треугольное неравенство (существует определение расстояния такое, что для любого

**1.1 Краткое описание документа**

Остальная часть бумаги организована следующим образом. В разделе 2 исследования, связанные с работой, и обобщены материалы BIRCH, в разделе 3 представлены некоторые справочные материалы. В разделе 4 представлены концепции функции кластеризации (CF) и CF tree, которые являются центральными для BIRCH. Подробности алгоритма BIRCH описаны в разделе 5, а предварительное исследование эффективности BIRCH представлено в разделе 6. Наконец, наши выводы и направления для будущих исследований представлены в разделе 7.

**2 Резюме соответствующих исследований**

Объединение в кластеры данных было изучено в Статистике [DH73, DJ80, Lee81, Mur83], Машинное обучение [CKS88, Fis87, Fis95, Leb87] и База данных [NH94, EKX95a, EKX95b] сообщества с различными методами и различными акцентами. Предыдущие подходы, основанные на вероятности (как и большинство подходов в машинном обучении) или на расстоянии (как большинство работ в статистике), недостаточно учитывают случай, когда набор данных может быть слишком большим, чтобы поместиться в основную память. В частности, они не признают, что проблема должна рассматриваться с точки зрения того, как работать с ограниченными ресурсами (например, памятью, которая обычно намного меньше, чем размер набора данных), чтобы выполнить кластеризацию как можно точнее, при сохранении низких затрат на I/0 .

**Вероятностные подходы**: Они обычно [Fis87, CKS88] делают предположение, что распределения вероятностей на отдельных в трибутах статистически независимы друг от друга. В реальности это далеко не так. ('orrelation between attributes exists, and::; ornPt, imes this come correlation is requirest we are seek. Представления вероятности кластеров имеют очень дорогое обновление и сохранение кластеров, особенно если атрибуты имеют большое количество значений, поскольку их сложности зависят не только от количества атрибутов, но и от количества значений для каждого атрибута. Связанная проблема заключается в том, что часто (например, [Fis87]) дерево, основанное на вероятности, которое строится для идентификации кластеров, не сбалансировано по высоте. Для искаженных входных данных это может привести к значительному снижению производительности.

**Подходы на основе расстояния**: они предполагают, что все точки данных даны заранее и их можно часто сканировать. Они полностью или частично игнорируют тот факт, что не все точки данных в наборе данных одинаково важны в отношении цели кластеризации, и что точки данных, которые близки и плотны, должны рассматриваться совместно, а не индивидуально. Они являются глобальными или полуглобальными методами при детализации точек данных. То есть для каждого решения кластеризации они одинаково проверяют все точки данных или все существующие в настоящее время кластеры независимо от того, насколько они близки или далеки друг от друга, и используют глобальные измерения, требующие сканирования всех точек данных или всех существующих кластеров. Следовательно, ни один из них не имеет линейной масштабируемости по времени со стабильным качеством.

Например, используя исчерпывающее перечисление (EE), существует приблизительно [DH73] способы -разделения набора из точек данных на подмножеств. Таким образом, на практике, хотя он может найти глобальный минимум, он неосуществим, за исключением случаев, когда и чрезвычайно малы. *Итеративная оптимизация* (IO) [DH73, KR90] начинается с начального раздела, затем пробует все возможные перемещения или перестановки точек данных из одной группы в другую, чтобы увидеть, улучшает ли такое перемещение или перестановка значение функции измерения. Он может найти локальный минимум, но качество локального минимума очень чувствительно к первоначально выбранному разделению, и худший случай сложности времени по-прежнему экспоненциальен. Иерархическая кластеризация () [DH73, KR90, Mur83] не пытается найти «лучшие» кластеры, а продолжает объединение ближайшей пары (или разделение самой дальней пары) объектов для образования кластеров. При разумном измерении расстояния наилучшей временной сложностью практического алгоритма является   
. Таким образом, он все еще не может хорошо масштабироваться с большим .

Недавно кластеризация была признана полезным методом пространственного анализа данных. [NH94] представляет CLARANS, который основан на рандомизированном поиске, и предлагает, что CLARANS превосходит традиционные алгоритмы кластеризации в статистике. В CLARANS дастер представлен его медоидом, или наиболее центрально расположенной точкой данных в кластере Процесс кластеризации формализуется как поиск графа, в котором каждый узел является апартионом, представленным множеством медоидов и два узла являются соседями, если они отличаются только одним медоидом. CLARANS начинается со случайно выбранного узла. Для текущего узла он проверяет максимум максимальное число соседей по соседству случайным образом, и если найден лучший сосед, он перемещается к соседу и продолжает; в противном случае он записывает текущий узел как локальный mintmum и перезапускается с новым случайно выбранным узлом для поиска другого локального минимума. CLARANS останавливается после того, как нумлокальный номер так называемых локальных минимумов был найден, и возвращает лучшее из них.

CLARANS имеет те же недостатки, что и вышеуказанный способ IO. Кроме того, он может не найти реальный локальный минимум из-за подстройки поиска, управляемой maxneighbor. Позже [EKX95a] и [EKX95b] предлагают методы фокусировки (основанные на -деревах) для улучшения способности CLARANS обращаться с объектами данных, которые могут находиться на дисках, путем (1) кластеризации выборки набора данных, который взят с каждой страницы данных -дерева; и (2) сосредоточение внимания на соответствующих точках данных для обновления расстояния и качества. Их эксперименты показывают, что время улучшается с небольшой потерей качества.

**2.1 Взносы БЕРЕЗЫ**

Важным вкладом является наша формулировка проблемы кластеризации таким образом, который подходит для очень больших наборов данных путем явного определения ограничений времени и памяти.

Кроме того, BIRCH имеет следующие преимущества по сравнению с предыдущими дистанционными подходами.

• BIRCH является локальным (в отличие от глобального) тем, что каждое решение о кластеризации принимается без сканирования всех точек данных или всех существующих в настоящее время источников данных. Он использует измерения, отражающие естественную близость точек, и в то же время может постепенно поддерживаться в процессе кластеризации.

• BIRCH использует наблюдение, что пространство данных обычно занято неравномерно, и, следовательно, не каждая точка данных одинаково важна для кластеризации. Плотная область точек рассматривается совместно как единое скопление. Точки в разреженных областях рассматриваются как выбросы и при необходимости удаляются.

• BIRCH в полной мере использует имеющуюся память для получения наилучших субкластеров (для обеспечения точности) при минимизации затрат на I/0 (для обеспечения эффективности). Процесс кластеризации и сокращения организован и характеризуется использованием в памяти, высотной и высоко занятой древовидной структуры. Благодаря этим особенностям время его работы линейно масштабируется.

• Если мы опустим необязательный этап 4 5, BIRCH — это инкрементный метод, который не требует заранее всего набора данных и сканирует набор данных только один раз.

**3 Предыстория**

Предположим, что читатели знакомы с терминологией векторных пространств, начнем с определения центроида, радиуса и диаметра для кластера. Если задано N d-мерных точек данных в кластере: где , центроид , радиус и диаметр пылеуловителя определяются как:

- среднее расстояние от точки элемента до центроида. - среднее попарное расстояние внутри кластера. Они представляют собой две альтернативные меры герметичности скопления вокруг центроида. Далее между двумя кластерами определим 5 альтернативных расстояний для измерения их близости.

Данные центроиды двух кластеров: и , центроидное евклидово расстояние и центроидное манхэттенское расстояние двух кластеров определяются как:

Данный d-мерные точки данных в кластере: где и точек данных в другом кластер: где , , , …, ,

среднее меж-кластерное расстояние , среднее внутрикластерное расстояние и расстояние увеличения дисперсии двух кластеров определяются как:

фактически является объединенного кластера. Для ясности рассмотрим , и как свойства одного кластера, а и как свойства между двумя кластерами и установим их отдельно. Пользователи могут дополнительно обрабатывать данные путем взвешивания или сдвига по различным измерениям, не влияя на относительное размещение.

**4 Функция кластеризации и дерево CF**

Концепции **функции кластеризации** и дерева лежат в основе инкрементальной кластеризации BIRCH. **Функция кластеризации** — это тройное суммирование информации, которую мы сохраняем о дастере. ***Определение 4.1*** Данные **N** d-мерных точек данных в кластере: где , вектор функции кластеризации (CF) кластера определяется как тройка: , где **N** - количество точек данных в кластере, LS - линейная сумма N точек данных, то есть: , а - квадратная сумма N точек данных, ч. т. д.

***Теорема 4.1*** (***Теорема аддитивности CF***): Предположим, что = (, , ) и = (, , ) являются **CF** векторами двух непересекающихся кластеров. Тогда **CF**-вектор кластера, который образуется объединением двух непересекающихся кластеров, равен:

Доказательство состоит из прямолинейной алгебры.

Из определения CF и теоремы аддитивности мы знаем, что CF векторы кластеров могут храниться и вычисляться инкрементно и точно при объединении кластеров. Также легко доказать, что учитывая CF-векторы кластеров, соответствующие и , а также обычные метрики качества (такие как взвешенный общий/средний диаметр дустеров) все могут быть легко вычислены.

Можно рассматривать кластер как набор точек данных, но только вектор хранится в виде сводки. Эта сводка эффективна не только потому, что хранит гораздо меньше, чем все точки данных в кластере, но и точна потому, что ее достаточно для расчета всех измерений, необходимых для принятия решений о кластеризации в BIRCH.

**4.1 Дерево CF**

Дерево CF — это сбалансированное по высоте дерево с двумя параметрами: коэффициентом ветвления B и порогом T. Каждый нелистовой узел содержит максимум B записей вида CF,. child;], где i = 1,2,..., B, «child» - указатель на его i-й дочерний узел, а CF - CF -подкластера, представленного этим потомком. Таким образом, nonleaf-узел представляет кластер, состоящий из всех подкластеров, представленных его записями. Листовой узел содержит максимум L записей, каждая из которых имеет вид (CF;], где i = 1,2,..., L. Кроме того, каждый листовой узел имеет два указателя, «prev» и «next», которые используются для цепочки всех листовых узлов вместе для эффективного сканирования. Конечный узел также представляет кластер, состоящий из всех подкластеров, представленных его записями. Но все записи в конечном узле должны удовлетворять пороговому требованию по отношению к пороговому значению T: диаметр (или радиус) должен быть меньше T.

Размер дерева является функцией Т. Чем больше T, тем меньше дерево. После того, как задана размерность d пространства данных, известны размеры листовых и неперекрытых элементов, то B и L определяются P. Так что P может изменяться для настройки производительности.

Такое CF-дерево будет создаваться динамически при вставке новых объектов данных. Он используется для направления новой вставки в правильный подкластер для целей кластеризации так же, как B + -дерево используется для направления новой вставки в правильное положение для целей сортировки. Дерево CF является очень компактным представлением набора данных, поскольку каждая запись в конечном узле не является единственной точкой данных, в которой находится подкластер (который поглощает множество точек данных с диаметром (или радиусом) под определенным порогом T). 4.2 Вставка в дерево CF

Теперь мы представляем алгоритм вставки записи в CF-дерево. Запись Uiven «Ent», она протекает следующим образом:

1. Определение соответствующего листа: Начиная с корня, он рекурсивно опускается по дереву CF, выбирая ближайший дочерний узел в соответствии с выбранной метрикой расстояния: D0,D1,D2,D3 или D4, как определено в разделе 3.

2. Изменение листа: Когда он достигает узла листа, он находит ближайший элемент листа, скажем L,, а затем проверяет, может ли L «поглощать» «Ent», не нарушая пороговое условие \*, если это так, вектор CF для L; обновляется, чтобы отразить это. Если нет. в лист добавляется новая запись для «Ent». Если на листе есть место для этой новой записи, мы закончим, в противном случае мы должны разделить узел листа. Разделение узла выполняется путем выбора самой дальней пары записей в качестве начальных значений и перераспределения оставшихся записей на основе ближайших критериев .

3. Изменение пути к листу: После вставки «Ent» в листок мы должны обновить информацию CF для

2То есть кластер, объединенный с «Ent» и L, должен удовлетворять пороговому условию. Следует отметить, что CF-вектор нового кластера может быть вычислен из CF-векторов для L и «Ent».

каждый элемент, не являющийся листом, на пути к листу. В отсутствие разделения это просто включает в себя добавление CF векторов, чтобы отразить добавление «Ent». Разбиение листа требует вставки в родительский узел новой записи без листа для описания вновь созданного листа. Если у родителя есть место для этой записи, на всех более высоких уровнях нам нужно только обновить CF-векторы, чтобы отразить добавление «Ent». В общем, однако, нам, возможно, придется разделить и родителя, и так далее вплоть до корня. Если корень разделен, высота дерева увеличивается на единицу.

4. Уточнение объединения: разбиение вызвано размером страницы, который не зависит от свойств кластеризации данных. При наличии искаженного порядка ввода данных это может повлиять на качество кластеризации, а также снизить использование пространства. Простой дополнительный этап объединения часто помогает облегчить эти проблемы: Предположим, что существует расщепление листа, и распространение этого расщепления останавливается в некотором узле А, то есть N, может вместить дополнительную запись, полученную в результате расщепления. Теперь мы сканируем узел N, чтобы найти две ближайшие записи. Если они не являются парой, соответствующей разбиению, мы пытаемся объединить их и соответствующие два дочерних узла. Если в двух дочерних узлах больше записей, чем может содержать одна страница, результат объединения разделяется снова. Во время повторного отображения, если одно из начальных чисел привлекает достаточно объединенных записей для заполнения страницы, мы просто помещаем остальные записи с другим начальным числом. Таким образом, если объединенные записи помещаются на одной странице, мы освобождаем пространство узла для последующего использования, создаем еще одно пространство входа в узле A, тем самым увеличивая использование пространства и откладывая будущие разделения; в противном случае мы улучшаем распределение записей у ближайших двух детей .

Поскольку каждый узел может содержать только ограниченное количество

записи из-за своего размера не всегда соответствуют естественному кластеру. Иногда два подкластера, которые должны были находиться в одном кластере, разбиваются на узлы. В зависимости от порядка ввода данных и степени перекоса также возможно, что два подкластера, которые не должны находиться в одном кластере, сохраняются в одном узле. Эти нечастые, но нежелательные аномалии, вызванные размером страницы, устраняются с помощью глобального (или полуглобального) алгоритма, который упорядочивает конечные записи по узлам (Фаза 3, обсуждаемая в Разделе 5). Другой нежелательный артефакт заключается в том, что если одна и та же точка данных вставляется дважды, но в разное время, две копии могут быть введены в отдельные конечные записи. Или, другими словами, иногда с искаженным порядком ввода точка может ввести листовую запись, которую она не должна была вводить. Эта проблема может быть решена с помощью дальнейших проходов уточнения данных (Фаза 4, обсуждаемая в Разделе 5).

**5 Алгоритм кластеризации BIRCH**

На рис. 1 представлен обзор БЕРЕЗЫ. Основной задачей Фазы 1 является сканирование всех данных и построение начального несемейного дерева CF с использованием заданного объема памяти.

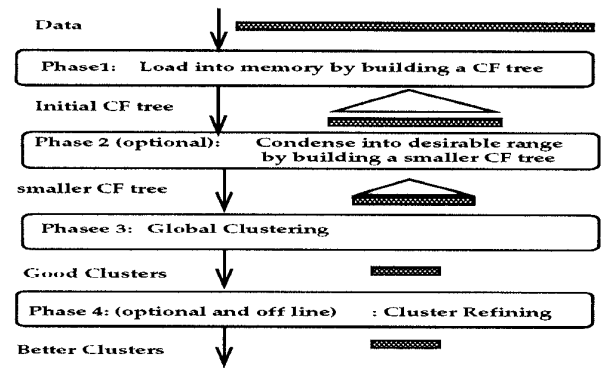


Рисунок 1: Обзор BIRCH

и повторное использование места на диске. Это дерево CF пытается максимально точно отразить информацию кластеризации набора данных при ограничении памяти. Поскольку переполненные точки данных сгруппированы как тонкие субкластеры, а разреженные точки данных удалены как выбросы, на этой стадии создается несемейная сводка данных. Подробная информация о Фазе 1 будет рассмотрена в Разделе 5.l. После Фазы 1 последующие вычисления на последующих фазах будут :

I.fast, поскольку (a) операции ввода-вывода не нужны, и (b) проблема кластеризации исходных данных сводится к меньшей проблеме кластеризации подкластеров в листовых записях;

2. точность, поскольку (а) множество выбросов исключается, и (б) оставшиеся данные отражаются с наибольшей детализацией, которая может быть достигнута при наличии доступной памяти ;

3. менее чувствительный к порядку, потому что листовые записи исходного дерева образуют порядок ввода, содержащий лучшую локальность данных по сравнению с произвольным порядком ввода исходных данных.

Этап 2 является необязательным. Мы наблюдали, что существующие глобальные или полуглобальные методы кластеризации, применяемые на этапе 3, имеют различные диапазоны входных размеров, в пределах которых они хорошо работают как с точки зрения скорости, так и качества. Таким образом, потенциально существует зазор между размером результатов Фазы 1 и входным диапазоном Фазы 3. Фаза 2 служит подушкой и перекрывает этот разрыв: Подобно Фазе 1, она сканирует листовые записи в начальном дереве CF, чтобы восстановить меньшее дерево CF, удаляя при этом больше выбросов и группируя переполненные подкластеры в более крупные.

Нежелательное влияние искаженного порядка ввода и разбиения, запускаемого размером страницы (раздел 4.2), заставляет нас быть неверными фактическим шаблонам кластеризации в данных. Это исправляется на этапе 3 с помощью глобального или полуглобального алгоритма для кластеризации всех конечных записей. Мы наблюдаем, что существующие алгоритмы кластеризации для набора точек данных могут быть легко адаптированы для работы с набором подкластеров, каждый из которых описывается своим CF вектором. Например, с известными CF векторами, (1) наивно, вычисляя центроид как репрезентативный

подкластера мы можем рассматривать каждый подкластер как единую точку и использовать существующий алгоритм без модификации; (2) или чтобы быть немного более сложным, WP может обрабатывать подкластер из n точек данных как его центроид, повторяющийся n раз, и слегка модифицировать существующий алгоритм для учета информации подсчета; (3) или чтобы быть общим и точным, мы можем применить существующий алгоритм непосредственно к подкластерам, потому что информация в их векторах CF обычно достаточна для вычисления большинства метрик расстояния и качества.

В этой статье мы адаптировали алгоритм агломеративной иерархической кластеризации, применив его непосредственно к подкластерам, представленным их CF векторами. Он использует метрику точного расстояния D2 или D4, которая может быть вычислена из CF векторов, во время всей кластеризации и имеет сложность O 2(N). Это также обеспечивает гибкость, позволяющую пользователю задавать либо требуемое количество кластеров, либо требуемое пороговое значение диаметра (или радиуса) для кластеров.

После фазы 3 мы получаем набор кластеров, который фиксирует основной шаблон распределения в данных. Однако незначительные и локализованные неточности могут существовать из-за редкой проблемы неправильного размещения, упомянутой в разделе 4.2, и того факта, что фаза 3 применяется к грубому обобщению данных. Этап 4 является необязательным и влечет за собой стоимость дополнительных проходов по данным для исправления этих неточностей и дальнейшей доработки очистных сооружений. Следует отметить, что до этого момента исходные данные сканировались только один раз, хотя информация о дереве и выбросах могла сканироваться несколько раз.

Фаза 4 использует центроиды кластеров, продуцируемых фазой 3, в качестве начальных значений, и перераспределяет точки данных к ближайшему начальному значению для получения набора новых кластеров. Это не только позволяет осуществлять миграцию точек, принадлежащих кластеру, но и гарантирует, что все копии данной точки данных попадают в один и тот же кластер. Фаза 4 может быть расширена с дополнительными проходами, если это желательно пользователем, и было доказано, что она сходится к минимуму [GG92]. В качестве бонуса, во время этого прохода каждая точка данных может быть помечена кластером, которому она принадлежит, если мы хотим идентифицировать точки данных в каждом кластере. Фаза 4 также предоставляет нам возможность отбрасывать выбросы. То есть точка, которая находится слишком далеко от своего ближайшего затравочного числа, может рассматриваться как выброс и не включается в результат .

**5.1 Пересмотр фазы 1**

На рис. 2 показаны детали фазы l. Она начинается с начального порогового значения, сканирует данные и вставляет точки в дерево. Если памяти не хватает до завершения сканирования данных, это увеличивает пороговое значение, перестраивает новое, меньшее дерево CF путем повторной вставки листовых записей старого дерева. После повторной вставки старых элементов листа сканирование данных (и вставка в новое дерево) возобновляется с точки, в которой они были прерваны.

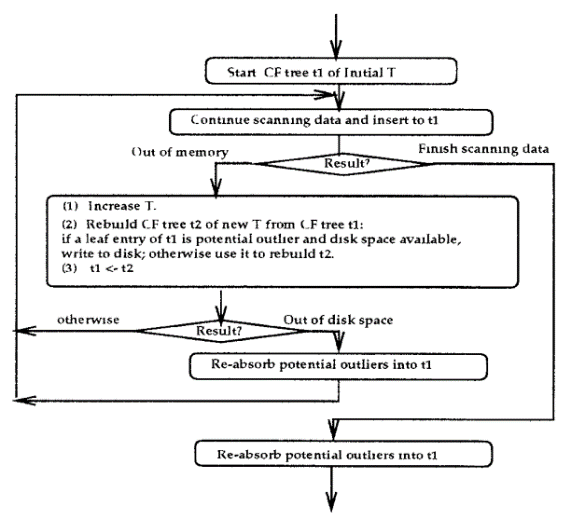


Рис. 2: Контролирующий поток фазы 1

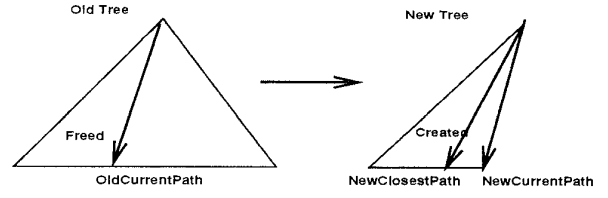


Рис. 3: Восстановление CF Tree

**5.1.1 Редуцируемость**

Предположим, что t - это CF-дерево порога T,. Его высота - h, а размер (количество узлов) - S,. Если T, + 1 2 T, мы хотим использовать все листовые записи t; для восстановления дерева CF. t, + 1 порога T + такой, что размер t, + не должен быть больше S, Далее следует алгоритм перестроения, а также теорема о последующей сводимости .

Предположим, что в каждом узле дерева CF t, записи помечены последовательно от 0 до , где - количество записей в этом узле, затем путь от записи в корне (уровень 1) к листовому узлу (уровень h) может быть однозначно представлен где - метка входа -го уровня на этом пути. Так естественно, путь если , и что листовой узел соответствует пути однозначно, и мы будем использовать путь и листовой узел взаимозаменяемо с текущего момента.

Алгоритм проиллюстрирован на рис. 3. При указанном выше порядке естественного пути он сканирует и освобождает старый путь дерева по пути, и одновременно создает новый путь дерева по пути. Новое дерево начинается с NULL, а «OldCurrentPath» начинается с крайнего левого пути в старом дереве. Для «Старого текущего пути» алгоритм

протекает, как показано ниже :

1. Создайте в новом дереве соответствующий ng «NewCurrentPath»: узлы добавляются в новое дерево точно так же, как в старом дереве, чтобы не было шансов, что новое дерево когда-либо станет больше старого дерева.

2. Вставка листовых записей в «OldCurrentPath» в новое дерево: с новым порогом каждая листовая запись в «OldCurrentPath» тестируется относительно нового дерева, чтобы проверить, может3 ли она поместиться в «NewClosestPath», который найден в верхнем клоуне с ближайшими критериями в новом дереве. Если да и «NewClosestPath» находится перед «NewCurrentPath», то он вставляется в «NewClosestPath», а пробел в «NewCurrentPath» остается доступным для последующего использования; в противном случае он вставляется в «NPwCurrentPath» без создания нового узла.

3. Свободное место в «OldCurrentPath» и «NewCurrntPath» После обработки всех конечных записей в «OldCurrentPath» ненужные узлы вдоль «OldC 'currentPath» могут быть освобождены. Также вероятно, что некоторые узлы $ вдоль «NewCurrentPath» пусты, потому что листовые записи, которые изначально соответствуют этому пути, теперь «вытесняются». В этом случае пустые узлы также могут быть освобождены .

4. «OldCurrentPath» zs устанавливается на следующий путь в старом дереве zf there Fxzsts one, и повторяются вышеописанные шаги.

Из шагов перестроения старые элементы листа -вставляются повторно, но новое дерево никогда не может стать больше старого дерева. Поскольку одновременно должны существовать только узлы, соответствующие «OldCurrentPath» и «NewCurrentPath», максимальное дополнительное пространство, необходимое для преобразования дерева, - h страниц. Таким образом, увеличивая порог, мы можем перестроить меньшее CF-дерево с ограниченной дополнительной памятью .

Теорема 5.1 (Теорема о редуцируемости:): Предположим, что мы перестраиваем CF дерево t, + 1 порогового Ta1 из CF дерева t, порога T, с помощью вышеприведённого алгоритма, и пусть S, и Sa + 1 являются размерами t, и t, + 1 соответственно. IfT, + 1 2 T, затем S + 1 £ S,, и преобразование от t, к t, + требует максимум h дополнительных страниц памяти, где h ws высоту t,

**5.1.2 Пороговые значения**

Хороший выбор порогового значения может значительно сократить количество перестроений. Поскольку начальное пороговое значение To динамически увеличивается, мы можем настроить его на слишком низкий уровень. Но если mitial T0 слишком высок, мы получим менее подробное дерево CF, чем возможно с доступной памятью. Так что To следует устанавливать консервативно. BIRCH по умолчанию устанавливает его на ноль; знающий пользователь может изменить это.

•3 Либо поглощается существующим элементом листа, либо создается как новый элемент листа без разделения

Предположим, что T, оказывается ему слишком маленьким, и у нас впоследствии заканчивается память после N, точки данных были отсканированы, и C, сформированы листовые записи (Pach, удовлетворяющий пороговому условию wrt. Т;). Исходя из части данных, которые мы отсканировали, и дерева, которое WP сформировал до сих пор, нам нужно оценить следующее пороговое значение T + 1. Эта оценка является сложной проблемой, и полное решение выходит за рамки этого документа. В настоящее время мы используем следующий эвристический подход :

l. Попытайтесь выбрать T + 1, чтобы A, + 1 = Min (2N, , N).

То есть, известно ли N, мы выбираем оценивать 'I' i + 1 максимум пропорционально данным, которые мы видели до сих пор.

2. Интуитивно мы хотим увеличить порог на основе somne меры объема. Существует два различных понятия объема, которые мы USP при оценке порога. Первый - средний объём, который определяется как V, = "где r - средний радиус корневого пылеуловителя в дереве CF, а d - размерность пространства. Интуитивно, это мера пространства, занятого частью данных, наблюдаемых до сих пор («след» видимых данных). Второе понятие объемного упакованного объема, которое определяется как V,, = (', + T, «где C' - количество листовых элементов и T» - максимальный объем листового элемента. Интуитивно это мера фактического объёма, занимаемого листовыми дустерами. Поскольку C, по сути, один и тот же всякий раз, когда у нас заканчивается память (поскольку мы работаем с фиксированным объёмом памяти), мы можем приблизить V на T,

Мы делаем предположение, что r растет с числом точек данных N;. Поддерживая запись r и количество точек AN;, мы можем оценить r, + 1 с помощью линейной регрессии наименьших квадратов. Мы определяем коэффициент расширения f = Max (1.0. "t ') и используем его в качестве эвристической меры роста площади данных. Использование Max мотивировано нашим наблюдением, что для большинства больших наборов данных наблюдаемый след становится постоянной довольно быстро (если только порядок ввода не искажен). Аналогично, делая предположение, что V растет линейно с A;, мы оцениваем 1/+ 1, используя линейную регрессию наименьших квадратов.

3. Мы проходим путь от корня к листу в дереве CF, всегда идя к ребенку с наибольшим количеством точек в «жадной» попытке найти самый переполненный листовой узел. Вычисляем расстояние (D, in) между ближайшими двумя записями на этом листе. Если мы хотим построить более сконденсированное дерево, разумно ожидать, что мы должны, по крайней мере, увеличить пороговое значение до D, nn, чтобы эти две записи могли слиться .

4. Умножили значение T1, полученное путём линейной регрессии, на коэффициент расширения f, и скорректировали его с помощью D, mn следующим образом: T,41 = Max (Dnn, f \* Ti + 1). Чтобы гарантировать, что пороговое значение растет монотонно, в очень маловероятном случае T + 1.

полученное таким образом меньше T, то выбираем 7 + 1 = T, + ("+) a. (Это эквивалентно предположению, что все точки данных равномерно распределены в d-мерной сфере, и на самом деле это просто грубое приближение, однако, оно редко вызывается .)

**5.1.3 Вариант обработки выбросов**

Опционально, мы можем использовать R байт дискового пространства для обработки выбросов, которые являются листовыми записями низкой плотности, которые считаются неважными wrt. общий шаблон кластеризации. При перестроении дерева CF путем повторной вставки старых элементов листа размер нового дерева уменьшается двумя способами. Во-первых, мы увеличиваем пороговое значение, тем самым позволяя каждому элементу листа «поглощать» точки rnorf. Во-вторых, мы рассматриваем некоторые листовые записи как потенциальные выбросы и записываем их на диск. Старый элемент листа 1 считается потенциальным выбросом, если он имеет «гораздо меньше» точек данных, чем среднее значение. «Гораздо меньше», конечно, еще одна эвристика.

Периодически пространство на диске может заканчиваться, и потенциальные выбросы сканируются, чтобы увидеть, могут ли они быть реабсорбированы в текущее дерево, не заставляя дерево расти в размерах. - Увеличение порогового значения или изменение распределения из-за новых данных, считанных после записи потенциального выброса, вполне может означать, что потенциальный выброс больше не квалифицируется как выброс. Когда все данные были отсканированы, потенциальные выбросы, оставшиеся в дисковом пространстве, должны быть просканированы, чтобы проверить, действительно ли они являются выбросами. Если потенциальный выброс не может быть поглощен при этом последнем шансе, он, весьма вероятно, является реальным выбросом и может быть удален.

Обратите внимание, что весь цикл - недостаточно памяти, запускающей перестроение дерева, недостаточно места на диске, что вызывает повторное поглощение! i <: 'rs, Ptc. - может повторяться несколько раз до полного сканирования набора данных. Эти усилия необходимо учитывать в дополнение к затратам на сканирование данных, чтобы точно оценить стоимость Фазы 1 .

**5.1.4 Опция Delay-Split**

Когда у нас заканчивается основная память, вполне может быть, что еще больше точек данных может поместиться в дерево currPnt CF без изменения порога. Однако некоторые точки данных, которые считывает WP, могут потребовать от нас разделения узла в дереве CF. Простая идея состоит в том, чтобы записать такие точки данных на диск (способом, подобным тому, как записываются выбросы), и продолжать считывать данные до тех пор, пока WP также не закончит место на диске. Преимущество такого подхода заключается в том, что в общем случае большее количество помп данных может поместиться в дереве до того, как нам придется перестраиваться.

**6 Исследования эффективности**

Мы представляем анализ сложности, а затем обсуждаем эксперименты, которые мы провели на BIRCH (и CLARANS) с использованием синтетических, а также реальных наборов данных.

**6.1 Анализ**

Сначала анализируем стоимость ЦП Фазы 1. Максимальный размер дерева - t.Чтобы вставить точку, нужно следовать пути от корня к листу, касаясь около 1 + logB t узлов. На каждом узле мы должны исследовать записи B, ища «ближайший»; стоимость одной записи пропорциональна аналитике d. Таким образом, стоимость вставки всех точек данных составляет O (d'N "B (+ logy {g)). Если мы должны перестроить дерево, пусть ES 'будет размером записи CF. Существует максимум! fs листовые записи для повторной вставки, поэтому стоимость повторной вставки листовых записей равна 0 (d \* ff., \* B (1 + logB t)). Количество раз, когда мы должны заново построить дерево зависит от наших пороговых эвристик. В настоящее время речь идет о log «», где значение 2 возникает из того, что мы никогда не оцениваем дальше, чем в два раза от текущего размера, а No - количество точек данных, загруженных в память с порогом T%. Таким образом, общая стоимость цп фазы 1 равна O (d'N + B (+ log "{) - + log" A + ad + g "B (+ log,)). Анализ стоимости ЦП Фазы 2 аналогичен и поэтому опущен.

Что касается ввода-вывода, мы сканируем данные один раз на фазе 1, а не на фазе 2. С включенными опциями обработки выбросов и разбиения задержек существуют некоторые затраты, связанные с записью записей выбросов на диск и считыванием их во время восстановления. Учитывая, что количество диска, доступного для обработки выбросов (и разбиения задержки), не превышает М, и что существует около журнала для повторной сборки, стоимость ввода-вывода Фазы 1 существенно не отличается от стоимости считывания в наборе данных. На основании вышеприведенного анализа, который на самом деле довольно пессимистичен, в свете наших экспериментальных результатов стоимость Фаз 1 и 2 должна линейно масштабироваться с N.

В фазе 3 нет ввода-вывода. Поскольку входные данные фазы 3 ограничены, стоимость ЦП фазы 3, следовательно, ограничена константой, которая зависит от максимального размера входных данных и глобального алгоритма, выбранного для этой фазы. Фаза 4 снова сканирует набор данных и помещает каждую точку данных в соответствующий кластер; затраченное время пропорционально N К. (Однако при новейших методах «ближайшего соседа» его можно улучшить GG92] быть почти линейным wrt. Н.)

**6.2 Генератор синтетических наборов данных**

Чтобы изучить чувствительность BIRCH к характеристикам широкого спектра входных наборов данных, мы использовали коллекцию синтетических наборов данных, сгенерированных генератором, который мы разработали. Формирование данных контролируется набором параметров, которые обобщены в таблице 1.

Каждый набор данных состоит из K кластеров из 2-d точек данных.

Кластер характеризуется количеством точек данных в нём (n), его радиусом (r) и центром (c). n находится в диапазоне [ni, n], а r находится в диапазоне [ri, ri] \*. После размещения кластеры охватывают диапазон значений в каждом 4 Заметим, что при n1 = "h количество точек фиксировано, а при ri = rμ радиус фиксирован.

|  |  |
| --- | --- |
| Образец | сетка, синус, случайный |
| Количество кластеров *A* | 4.. 256 |
| *n* (нижний n) | 0 .. 2500 |
| *nμ* (Higher n) | 50. 2500 |
| *ri* (нижний r) | о. v2 |
| *r%* (выше r) | *v2* • v32 |
| Множитель расстояния *кг* | 4 ( только сетка) |
| Количество циклов *nc* | 4 *(только синус)* |
| Уровень шума *r,* (%) | 0 .. 10 |
| Порядок ввода *o* | рандомизированные, упорядоченные |

Таблица 1. Параметры формирования данных и их значения или диапазоны экспериментировали размерность. Мы называем эти диапазоны «обзором» набора данных.

Местоположение центра каждого кластера определяется параметром шаблона. *Сетка* трех шаблонов, синусоидальная и *случайная* - в настоящее время поддерживаются генератором. При использовании шаблона сетки центры кластера размещаются на сетке -. Расстояние между центрами соседних кластеров в одной строке/столбце управляется , и устанавливается равным . Это приводит к обзору   
в обоих измерениях. Синусоидальный шаблон помещает центры кластера на кривую синусоидальной функции. K кластеров делятся на nc-группы, каждая из которых помещается на различный цикл синусоидальной функции. Положение центра кластера равно , тогда как положение y равно  . Обзор набора синусоидальных данных, поэтому и [] на x и y соответственно. Случайный шаблон помещает кластерные центры случайным образом. Обзор набора данных представляет собой на обоих измерениях, поскольку местоположения центров x и y случайным образом распределены в пределах диапазона

Как только характеристики каждого кластера -определены, точки данных для кластера генерируются в соответствии с двумерным независимым нормальным распределением, среднее значение которого равно центру c, и дисперсия которой в каждом измерении - r;. Обратите внимание, что из-за свойств нормального распределения, максимальное расстояние между точкой в дустере и центром не ограничено. Другими словами, точка может быть произвольно далека от -своего кластера. Так что точка данных, которая принадлежит кластеру A, может быть ближе к центру кластера B, чем к центру A, и мы называем такие точки «посторонними».

В дополнение к кластеризованным точкам данных, шум в виде точек данных, равномерно распределенных по всему обзору набора данных, может быть добавлен к набору данных. Параметр r управляет процентом точек данных в наборе данных, которые считаются шумами.

Размещение точек данных в наборе данных контролируется параметром порядка o. При использовании рандомизированной опции точки данных всех кластеров и шум рандомизируются по всему

///

|  |  |  |
| --- | --- | --- |
| Глобальный | Память (М) | 80x1024 байт |
|  | Диск (R/ | 20% М |
|  | Distance def. | D2 |
|  | Quality def. | (D) |
|  | Пороговое значение def. | порог для *D* |
| Фазель | Начальный порог | 0.0 |
|  | Разделение задержки | на |
|  | Размер страницы (P) | 1024 байта |
|  | Обращение с выбросами | на |
|  | Outlier def. | Листовая запись, которая |
|  |  | содержит < 25% |
|  |  | среднее число |
|  |  | точек на лист |
|  |  | вход |
| Фаза | Диапазон ввода | 1000 |
|  | Алгоритм | Адаптированные УВ |
| Phase4 | Пропуск на доработку | 1 |
|  | Отбросить-выбросить | прочь |
|  | Outlier def. | Точка данных, |
|  |  | Евклидово расстояние |
|  |  | к ближайшему семени |
|  |  | больше, чем в два раза |
|  |  | радиуса |
|  |  | что кластер |

Таблица 2: Параметры BIRCH и их значения по умолчанию.

В то время как при выборе упорядоченной опции точки данных кластера размещаются вместе, кластеры располагаются в порядке их генерации. и шум размещают в конце.

**6.3 Параметры и настройки по умолчанию**

BIRCH способен работать при различных настройках. В таблице 2 приведены параметры BIRCH, их действующие объемы и значения по умолчанию. Если явно не указано иное, эксперименты проводятся в этой настройке по умолчанию.

M был выбран на 80 кбайт, что составляет около 5% от размера набора данных в базовой рабочей нагрузке, используемой в наших экспериментах. Поскольку дисковое пространство (R) используется только для выбросов, мы предполагаем, что R < M и множество R = 20% от M. Эксперименты по эффектам 5 метрик расстояния в первых 3 фазах [ZRL95] показывают, что (1) использование D3 в фазах I и 2 приводит к гораздо более высокому конечному порогу и, следовательно, дает кластеры худшего качества; (2) однако, нет отличительного различия в производительности среди других. Поэтому мы решили выбрать D2 по умолчанию. Следуя традиции Статистики, мы выбираем «средневзвешенный диаметр» (обозначаемый как D) в качестве измерения качества. Чем меньше D, тем лучше качество. Пороговое значение определяется как пороговое значение для диаметра кластера по умолчанию.

В **фазе 1** начальное пороговое значение по умолчанию равно 0. Основываясь на исследовании того, как размер страницы влияет на производительность [Z RL95], мы выбрали P - 1024. Опция delay-split включена, так что при заданном пороговом значении дерево CF принимает больше точек данных и достигает более высокой емкости. Вариант внешнего манипулирования включен таким образом, что BIRCH mu удаляет выбросы и концентрируется на плотных местах с заданным количеством ресурсов. Для простоты лечим листок запись, число точек данных которой составляет менее четверти среднего значения, в качестве выброса .

В **фазе 3** большинство глобальных алгоритмов могут достаточно хорошо обрабатывать 1000 объектов. Поэтому по умолчанию используется диапазон ввода 1000. Мы выбрали адаптированный алгоритм HC 'для использования здесь. Мы решили дать Фазе 4 доработать разрушители только один раз с его опцией отбрасывания-выброса, чтобы все точки данных были подсчитаны в измерении качества для справедливых сравнений .

**6.4 Базовая производительность рабочей нагрузки**

Первый набор экспериментов заключался в оценке способности BIRCH создавать различные большие наборы данных. Все времена представлены на втором месте в этой статье. Были использованы три синтетических набора данных, по одному для каждого шаблона. В таблице 3 представлены настройки генератора для них. Средневзвешенные диаметры фактических кластеров «D» также включены в таблицу.

Рис. 6 визуализирует действительные кластеры DS1, построив плоттер в виде окружности, центром которой является центроид, радиусом - радиус кластера, а меткой - количество точек в кластере. Кластеры DS1 BIRCH представлены на рис. 7. Мы видим, что BIR ('H dusters arP очень похож на фактические кластеры с точки зрения местоположения. количество точек и радиусы. Максимальная и средняя разница между центроидами фактического кластера и соответствующего ему кластера BIRCH составляет 0,17 и 0,07 соответственно. Количество точек в кластере BIRCH не более чем на 4% отличается от соответствующего собственно кластера. Радиусы дустеров BIRCH (в диапазоне от 1,25 до 1 ,40 со средним значением 1,32) сцеплены с радиусами собственно дустеров (1,41). Обратите внимание, что все радиусы BIRCH меньше фактических радиусов. Это связано с тем, что BIRCH назначает «посторонних» настоящих дастеров надлежащему BIRCH-дастеру. К подобным выводам можно прийти, проанализировав визуальные представления DS2 и DS3 (но опущенный здесь ключ к отсутствию места).

Как обобщено в таблице 4, для кластеризации 100 000 точек данных каждого набора данных BIRCH потребовалось менее 50 секунд (на рабочей станции HP 9000/720). Шаблон набора данных почти не повлиял на время кластеризации. В таблице 4 также представлены результаты рабочих характеристик для трех дополнительных наборов данных - DS1o, DS20 и DS30 - которые соответствуют DSI, DS2 и DS3 соответственно, за исключением того, что параметр o генератора установлен в порядок. Как показано в таблице 4, изменение порядка точек данных почти не повлияло на производительность BIRCH.

**6.5 Чувствительность к параметрам**

Мы изучили чувствительность работы BIRCH к изменению значений некоторых параметров. Из-за нехватки места здесь мы можем лишь представить некоторые основные выводы (подробнее см. [ZRL95]).

5С этого момента мы называем кластеры, генерируемые генератором, как «фактические кластеры», тогда как кластеры, идентифицированные BIRCH, как «BIRCH кластеры».

///

|  |  |  |
| --- | --- | --- |
| DS1 | сетка, *K* = 100, *nμ* = *nμ* = 1000*, r*; *= r* "= *2*, kg *=* 4, rn *=* 0*%, o = рандомизировано* | 2.00 |
| DS2 | sine, K = 100, *n;* = *n* = 1000*, rt = r* "= *V2,* ne *= 4*, r, = *0*%, *o = рандомизировано* | 2.00 |
| DS3 | *случайные, K = 100, nμ = 0,7 μ = 2000*, rμ *= 0,* rμ *= 4, rn* = *r, = 0*%, *o = рандомизированные* | 4.18 |

Таблица 3. Наборы данных, используемые в качестве базовой рабочей нагрузки

**Начальный порог**: (1) Производительность BIRCH стабильна до тех пор, пока начальный порог не будет чрезмерно высоким. набор данных. (2) 7% = 0,0 хорошо работает с небольшим дополнительным временем работы. (3) Если пользователь действительно знает хорошее Кому, то он может быть вознагражден, сэкономив до 10% времени .

**Размер страницы P**: В фазе 1 меньший (больший) P имеет тенденцию уменьшать (увеличивать) время работы, требует более высокого (нижнего) порога окончания, производит меньше (больше), но «более крупные (мелкие)» листовые записи и, следовательно, ухудшает (улучшает) качество. Однако с уточнением в Фазе 4 эксперименты предполагают, что от P - 256 до 4096, хотя качества в encl Фазы 3 различны, конечные качества после уточнения почти одинаковы.

**Опции выбросов**: BIRCH был протестирован на «шумных» наборах данных со всеми опциями выбросов, включенными и удален.Результаты показывают, что при всех опциях выбросов, BIRCH не медленнее, но быстрее, и в то же время, его качество намного лучше .

**Объем памяти**: В фазе 1, когда размер памяти (или максимальный размер дерева) увеличивается, время работы увеличивается из-за обработки большего дерева на перестроенное, но незначительно, потому что это делается в памяти; (2) генерируется больше, но более мелких субкластеров для питания следующей фазы и, следовательно, приводит к лучшему качеству; (3) неточность, вызванная недостаточным объемом памяти, может быть в некоторой степени компенсирована уточнениями фазы 4. Другим словом, BIRCH может пойти на компромисс между памятью и временем для достижения аналогичного конечного качества.

**6.6 Масштабируемость по времени**

Для проверки масштабируемости BIRCH используются два различных способа увеличения размера набора данных.

**Увеличение количества точек на кластер**: Для каждого из и мы создаем диапазон наборов данных, сохраняя настройки генератора одинаковыми, за исключением изменения ni и na для изменения n, и, следовательно, N. Время работы для первых 3 фаз, а также для всех 4 фаз строится в зависимости от размера набора данных на рис. 4. Показано, что оба они растут линейно.   
 последовательно для всех трех моделей.

**Увеличение количества кластеров**: Для каждого из DS1, DS2 и DS3 мы создаем диапазон наборов данных, сохраняя настройки генератора одинаковыми, за исключением изменения для изменения . Время работы для первых фаз, а также для всех 4 фаз строится в зависимости от размера набора данных на рис. 5. Поскольку и , и растут, и сложность Фазы 4 теперь составляет (может быть улучшена, чтобы быть почти линейной в будущем), общее время не является точно линейным

///

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| DS1 | 47.1 | 1.87 | DS1 | 47.4 | 1.87 |
| DS2 | 47.5 | 1.99 | DS20 | 46.4 | 1.99 |
| DS3 | 49.5 | 3.39 | DS3 | 48.4 | 3.26 |

Таблица 4: Производительность BIRCH при базовой рабочей нагрузке . Время, и порядок ввода

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| DS1 | 839.5 | 2.11 | DS1o | 1525.7 | 10.75 |
| DS2 | 777.5 | 2.56 | DS20 | 1405.8 | 179.23 |
| DS3 | 1520.2 | 3.36 | DS3 | 2390.5 | 6.93 |

Таблица 5. Производительность CLARANS при базовой рабочей нагрузке wrt. Время, D и порядок ввода

. . Однако снова подтверждается, что время работы первых 3 фаз растет линейно. последовательно для всех трех моделей.

**6. 7 Сравнение данных BIRCH и CLARANS**

В этом эксперименте WP сравнивает производительность CLARANS и BIRCH с базовой нагрузкой. Во-первых, CLARANS предполагает, что памяти достаточно для хранения всего набора данных, поэтому ей требуется гораздо больше памяти, чем BIRCH. Чтобы CLARANS останавливался после приемлемого времени работы, мы устанавливаем его максимальное соседнее значение как большее 50 (вместо 250) и 1,25% , но не более 100 (вновь введенный верхний предел, рекомендованный ). Его числовое значение по-прежнему равно 2. На рис. 8 представлены кластеры CLARANS для . Сравнивая их с фактическими кластерами для , мы можем наблюдать, что: (1) Шаблон расположения центров кластера искажен. (2) Количество точек данных в кластере CLARANS может на целых 57% отличаться от количества в фактическом кластере. (3) Радиусы кластеров CLARANS в основном варьируются от 1,15 до 1,94 при среднем значении 1,44 (больше, чем у реальных кластеров, 1,41). Подобное поведение можно наблюдать за визуализацией кластеров CLARANS для DS2 и DS3 (но здесь опущено указание на отсутствие места).

В таблице 5 приведены результаты работы CLARANS. Для всех трех наборов данных базовой рабочей нагрузки (1) CLARA NS по меньшей мере   
в 15 раз медленнее, чем BIRCH, и чувствителен к шаблону набора данных. (2) Значение D для кластеров CLARANS намного больше, чем для кластеров BIRCH. (3) Результаты для DS1o, DS20 и DS30 показывают, что когда точки данных упорядочены, время и качество CLARANS резко ухудшаются. В заключение, для базовой нагрузки BIRCH использует гораздо меньше памяти, но быстрее, точнее и менее чувствителен к порядку по сравнению с CLARANS.

///

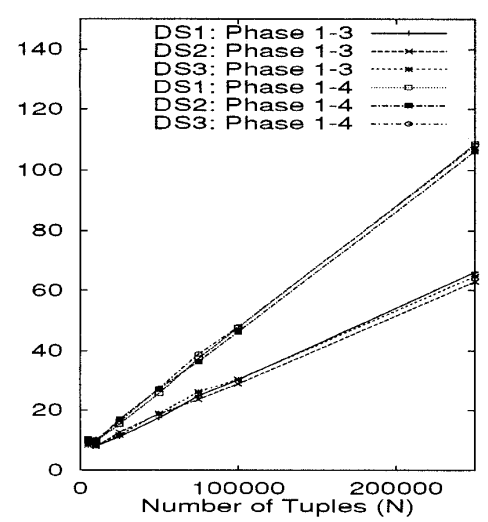


Рис. 4. Масштабируемость wrt при увеличении

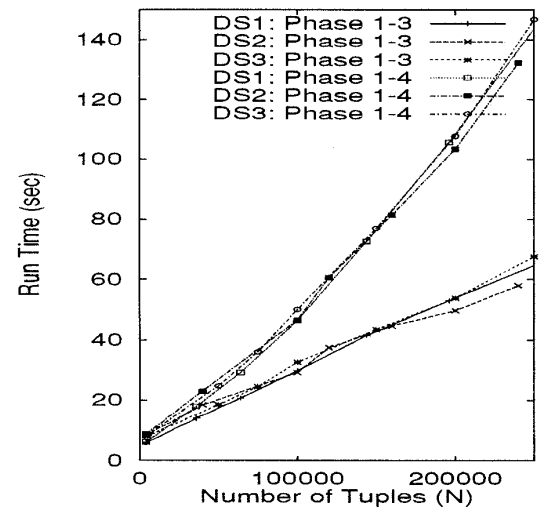


Рисунок 5. Масштабируемость wrt при увеличении K

**6.8 Приложение к реальным наборам данных**

Для фильтрации реальных изображений использовался BIRCH. На рис. 9 представлены два смоделированных изображения деревьев с переменным облачным небом в качестве фона, сделанные на двух разных длинах волн. Верхний - в ближней инфракрасной полосе (NIR), а нижний - в видимой полосе длин волн (VIS). Каждое изображение содержит пикселей, и каждый пиксель фактически имеет пару значений яркости, соответствующих NIR и VIS. Почвоведы получают сотни таких пар изображений и пытаются сначала фильтровать деревья с заднего плана, а затем фильтровать деревья в солнцезащитные листья, тени и ветви для статистического анализа.

Мы применили BIRCH пар значений (NIR, VIS) для всех пикселей в изображении ( 2-d кортежи), используя 400 килобайт памяти (около 5% размера набора данных) и 80 килобайт дискового пространства (около 20% объема памяти),

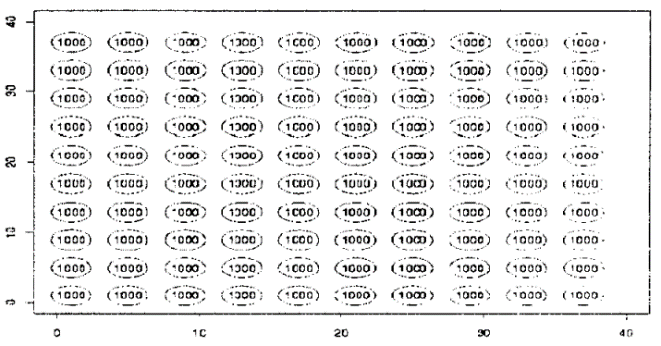


Рисунок 6. Фактические кластеры

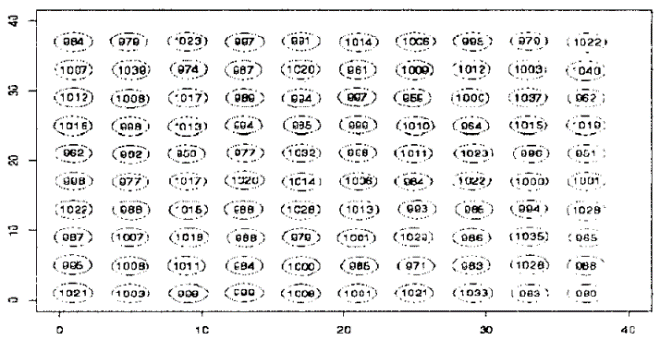
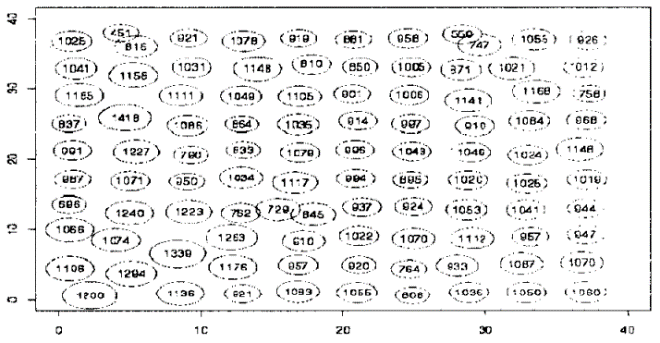


Рисунок 7: BIRCH-кластеры



и одинаковое взвешивание значений NIR и VIS. Мы получили 5 скоплений, которые соответствуют (1) очень яркой части неба, (2) обычной части неба, (3) облакам, (4) солнечным листьям (5) ветвям деревьев и теням на деревьях. Этот шаг занял 284 секунды.

Однако ветви и тени были слишком похожи, чтобы их можно было отличить друг от друга, хотя мы могли бы отделить их от других кластерных категорий. Так мы вытащили часть данных, соответствующую (5) (146707 2-d кортежи) и снова использовали BIRCH. Но на этот раз, (1) NIR. был взвешен в 10 раз тяжелее, чем VIS, так как мы наблюдали, что ветви и тени легче отличить от NIR-изображения, чем от VIS-изображения; (2) BIRCH закончился с более тонким порогом, потому что он обрабатывал меньший набор данных с объемом памяти. Два кластера, соответствующие ветвям и теням, были получены за 71 секунду.

На рис. 10 показаны части изображения, соответствующие солнечные листья, ветви деревьев и тени на деревьях, полученные кластеризацией с помощью BIRCH.

///



Рисунок 9. Изображения, полученные в NIR и VIS



Рисунок 10: Солнечные листья, ветви и тени

Визуально мы видим, что это удовлетворительная фильтрация исходного изображения в соответствии с намерением пользователя.

**7 Резюме и будущие исследования**

BIRCH — это метод кластеризации для очень больших наборов данных. Это делает проблему большой кластеризации прослеживаемой, концентрируясь на плотно занятых частях и используя компактную сводку. Он использует измерения, которые фиксируют естественную близость данных. Эти измерения могут храниться и обновляться с шагом в дереве с балансировкой по высоте. BIRCH может работать с любым заданным объемом памяти, а сложность ввода-вывода составляет чуть больше одного сканирования данных. Экспериментально показано, что BIRCH очень хорошо работает на нескольких больших наборах данных и значительно превосходит CLARA NS по качеству, скорости и чувствительности к порядку.

Правильная настройка параметров важна для эффективности работы BIRCH. В ближайшее время сконцентрируемся на

(1) изучение более разумных способов динамического увеличения порога, (2) динамическая корректировка критериев выбросов, (3) более точные измерения качества и (4) параметры данных, которые являются хорошими показателями того, насколько хорошо BIRCH может работать. Мы изучим архитектуру BIRCH для возможностей параллельного выполнения, а также интерактивного обучения. В качестве инкрементного алгоритма BIRCH сможет считывать данные непосредственно с ленточного накопителя или из сети путем согласования скорости кластеризации со скоростью считывания данных. Мы также рассмотрим, как использовать полученную информацию кластеризации для решения таких проблем, как хранение или оптимизация

**Ссылки**

[CKS88] Питер Чизман, Джеймс Келли, Мэтью Селф и др., Авто Класс: Байесовская система классификации, проф. 5-го инт. Conf. on Machine Learning, Morgan Kaufman, Jun. 1988 .

[DH73] Ричард Дуда и Питер Э. Харт, классификация шаблонов и анализ сцен, Wiley, 1973.

[DJ80] Р. Дюбс и А. К. Джейн, Clustering Methodolistics in Exploratory Data Analysis Advances in Computers, Edited by M.C. Yovits, Vol. 19, Academic Press, New York, 1980.

[EKX95a] Мартин Эстер, Ханс-Петер Кригель и Сяовэй Сюй, интерфейс базы данных для кластеризации в больших пространственных базах данных, процесс 1-го Международного конгресса по обнаружению знаний и анализу данных, 1995.

[EKX95b] Мартин Эстер, Ханс-Петер Кригель и Сяовэй Сюй, «Открытие знаний в больших пространственных базах данных: методы фокусировки для эффективной идентификации классов», Proc. of 4th Int '! Симпозиум по большим пространственным базам данных, Портленд, Мэн, США, 1995.

[Fis87] Дуглас Х. Фишер, получение знаний посредством инкрементальной -концептуальной кластеризации, машинное обучение, 2 ( 2), 1987

[Fis95] Дуглас Х. Фишер, Итеративная оптимизация и упрощение -иерархических кластеров, Технический отчет CS-95–01, Департамент компьютерных наук, Университет Вандербильта, Нэшвилл, TN 37235.

[GG92] A. Gersho and R. Gray, Vector quantization and signal compression, Boston, Ma.: Kluwer Academic Publishers, 1992.

[KR90] Леонард Кауфман и Питер Дж. Руссью, Finding Groups in Data - An Introduction to Cluster Analysis, Wiley Series in Probability and Mathematical Statistics, 1990. [Leb87] Michael Lebowitz, Experiments with Incremental Concept Formation: UNIMEM, Machine Learning, 1987.

[Lee81] R.C.T.Lee, Clustering analysis and its applications, Advances in Information Systems Science, Edited by J.T.Toum, Vol. 8, pp. 169-292, Plenum Press, New York, 1981.

[Mur83] F. Murtagh, A Survey of Recent Advances in Hierarchical Clustering Algorithms, The Computer Journal, 1983.

[NH94] Raymond T. Ng and Jiawei Han, Efficient and Effective Clustering Methods for Spatial Data Mining, Proc. of VLDB, 1994.

[Ols93] Кларк Ф. Олсон, параллельные алгоритмы иерархической кластеризации, технический отчет, отдел компьютерных наук, Калифорнийский университет в Беркли, Dec.,1993 .

[ZRL95] Тянь Чжан, Рагху Рамакришнан и Мирон Ливни, BIRCH: Эффективный метод кластеризации данных для очень больших баз данных, Технический отчет, Отдел компьютерных наук, Юнв. Висконсин-Мэдисон, 1995.