САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

Чередник Кирилл Евгеньевич

Сравнительный анализ алгоритмов расщепления вершин в R-дереве

Дипломная работа

Допущена к защите.
Зав. кафедрой:
д. ф.-м. н., профессор Терехов А. Н.

Научный руководитель:
ассистент Чернышев Г. А.

Рецензент:
д. ф.-м. н., профессор Новиков Б. А.

Санкт-Петербург
2014

SAINT-PETERSBURG STATE UNIVERSITY
Mathematics & Mechanics Faculty

Software Engineering Chair

Kirill Cherednik

Comparative analysis of R-tree split algorithms

Graduation Thesis

Admitted for defence.
Head of the chair:
professor Andrey Terekhov

Scientific advisor:
assistant professor George Chernishev

Reviewer:
professor Boris Novikov

Saint-Petersburg
2014

Оглавление

Введение

При работе с большими объемами многомерных данных часто возникает задача организации эффективного доступа к этим данным. Среди областей, в которых эта задача особенно актуальна, можно выделить геоинформационные системы. Одним из возможных подходов к решению является использование механизмов пространственного индексирования [6].

Перечислим типичные запросы к пространственному индексу:

Запрос на диапазон (Range query) — вернуть все объекты принадлежащие данному диапазону;

К ближайших соседей (KNN query) — вернуть K ближайших к данной точке объектов;

Точечный запрос (Point query) — вернуть все объекты, находящиеся в данной точке.

Среди структур данных используемых в области пространственного индексирования очень популярны иерархические структуры данных, такие как KD-дерево, Quad-дерево, и др. Они показывают хорошую производительность на данных малой размерности. Среди них стоит особо отметить семейство R-деревьев [14].

R-дерево [8] является обобщением B^+-дерева [13] на многомерный случай, посредством использования n-мерных прямоугольников для ограничения объектов в поддереве. Оно реализовано во многих СУБД индустриального уровня, таких как Oracle, PostgreSQL, SQLite, MySQL и др. В данной работе упор сделан на структуры данных, основанные на R-дереве, ввиду их большой популярности в сообществе разработчиков СУБД.

Одним из ключевых моментов, влияющих на производительность R-дерева, является алгоритм, решающий задачу расщепления переполненной вершины при вставке новой записи, который, наряду с алгоритмом выбора поддерева для вставки нового элемента, определяет распределение записей по его вершинам. Было предложено несколько алгоритмов расщепления вершин. Помимо этого были описаны различные вариации R-дерева: R*-дерево [15], Revised R*-дерево (RR*-дерево) [3], Гильбертово

R-дерево [10], в которых были затронуты и другие аспекты R-дерева. Однако каждая новая работа проводит сравнения далеко не со всеми существующими перспективными альтернативами, сравнения проводятся с учетом разных критериев, на различных платформах и нагрузках. Поэтому возникает проблема выявления наилучшего решения.

Использование индексов в параллельных СУБД накладывает дополнительные требования к структурам данных. Одним из самых важных требований является эффективная реализация механизма обеспечения корректности параллельного доступа к данным.

Стоит отметить, что снижение цен на оперативную память способствовало росту интереса к индексам, работающим полностью в оперативной памяти. На рынке представлены OLTP системы, размещающие таблицы с данными в оперативной памяти, например: MariaDB, SQLite, MS SQL Server 2014. Из этого следует то, что исследование индексов в оперативной памяти является актуальной задачей. Данная работа рассматривает именно такие индексы.

Наличие множества различных структур данных для индексирования, трудность реализации для каждой из них механизма обеспечения параллельного доступа и поддержки транзакций, а также доказательства их корректности привели к появлению обобщенного дерева поиска (GiST) [9]. Он позволяет абстрагировать дерево поиска, реализовав для него вышеупомянутые механизмы.

Существует несколько программных реализаций GiST (PostgreSQL, LibGiST, и др.) - как промышленных, так и академических, но многие из них не подходят для данной работы из-за неполной функциональности: отсутствие поддержки многопоточности, транзакций, таблиц, располагающихся в оперативной памяти. По этой причине в данной работе используется разрабатываемый на математико-механическом факультете СПбГУ[1], прототип системы многомерного индексирования, который включает в себя реализацию структуры GiST и поддерживает уровень изоляции транзакций *read committed*.

[1]работа частично поддержана грантом РФФИ №12-07-31050

Постановка задачи

Целью данной работы является изучение того, как выбор алгоритма расщепления вершин в R-дереве влияет на производительность системы пространственного индексирования.

В связи с этим были поставлены следующие задачи:

1. провести обзор подходов к решению задачи расщепления вершин в R-дереве и критериев их сравнения;
2. проанализировать возможность встраивания изученных деревьев в структуру данных GiST;
3. при отсутствии возможности непосредственного встраивания дополнительных деревьев поиска в структуру данных GiST, провести его модификацию, не влияющую при этом на алгоритмы поддержки многопточности;
4. провести экспериментальное сравнение выбранных структур данных для пространственного индексирования на единой платформе и тестовом наборе данных.

1. Обзор семейства R-деревьев

1.1. R-дерево

Согласно [8] R-дерево — это древовидная структура данных, заданная парой (m, M), где $m \leq \frac{M}{2}$ со следующими свойствами:

- каждая листовая вершина (если она не корень) содержит от m до M записей;
- каждая запись в листовой вершине представлена в виде пары (mbr, oid), где mbr это минимальный ограничивающий прямоугольник, содержащий данный объект, а oid — идентификатор объекта;
- каждая внутренняя вершина (если она не корень) содержит от m до M записей;
- каждая запись во внутренней вершине представляется в виде пары (mbr, p), где p — указатель на дочернюю вершину, а mbr это минимальный ограничивающий прямоугольник, содержащий все записи соответствующего ребенка;
- минимальное количество записей в корне — 2, за исключением случая, когда корень является листовой вершиной;
- дерево сбалансировано.

Пример данных и соответствующего R-дерева, представлены на Рис. 1. Здесь данные — прямоугольники $E - O$, а $A - D$ — минимальные ограничивающие прямоугольники.

Можно понимать R-дерево, как обобщение B^+-дерева [13]:

- данные так же хранятся в листьях;
- оба дерева сбалансированы;
- внутренние вершины содержат ограничивающие прямоугольники, которые можно считать обобщением интервала, используемого для поиска в B^+-дереве.

Основные же отличия R-дерева от B^+-дерева перечислены ниже.

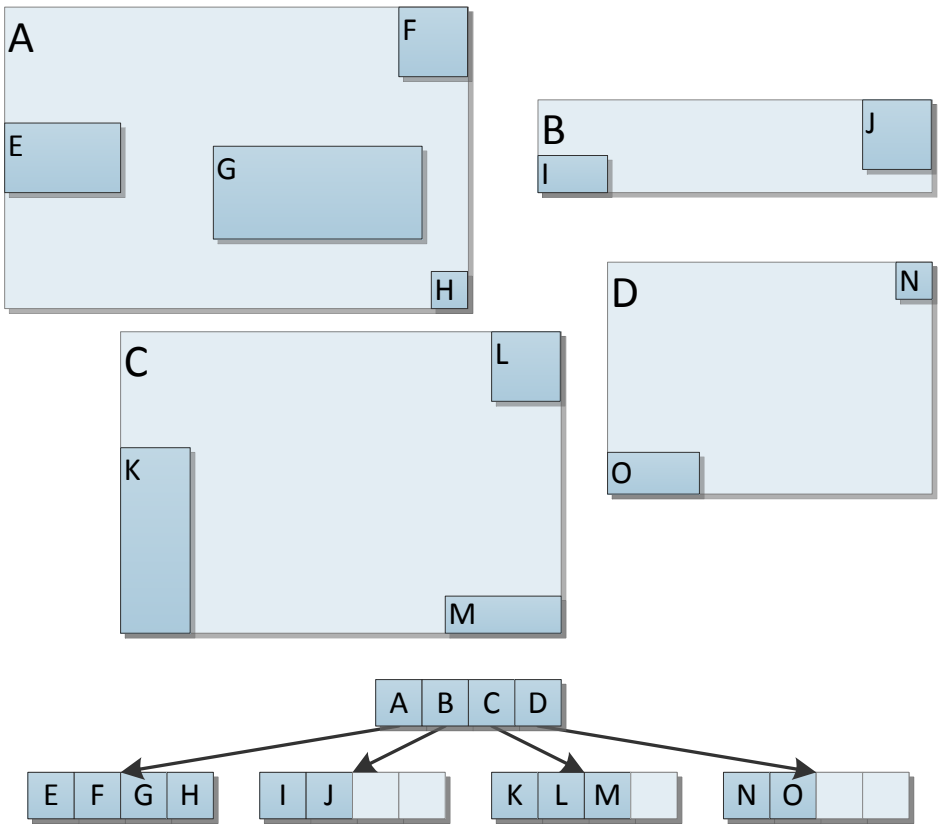

Рис. 1: R-дерево — пример

- Для нахождения записей с данным ключом может потребоваться проход по нескольким веткам. Это вызвано возможностью пересечения ограничивающих прямоугольников.

- Расщепление переполненной вершины — нетривиальная задача. Неудачный выбор алгоритма может привести к неоптимальной структуре дерева, что приведет к серьезному снижению производительности системы.

- В оригинальном R-дереве, описанном в статье [8], листья не образуют связаный упорядоченный список.

Одним из основных факторов, влияющих на производительность R-дерева, является качество алгоритм расщепления вершин [15]. Формально задача “хорошего’’ расщепления формулируется следующим образом: разделить множество прямоугольников на две группы так, чтобы минимизировать некоторую целевую функцию. Выбор целевой функции определяет критерии качества расщепления вершин.

С момента публикации оригинальной статьи про R-дерево было проведено немало исследований в этой области, что привело к появлению множества его адаптаций,

большинство из которых делают упор на алгоритме расщепления вершин. В данной работе рассматриваются самые популярные из этих адаптаций.

Зачастую внимание уделяется следующим критериям, определяющим качество расщепления вершины [15]:

1. площадь прямоугольника, соответствующего вершине, минус общая площадь прямоугольников, соответствующих её дочерним вершинам;
2. площадь пересечения прямоугольников вершин, получившихся в результате разделения;
3. степень заполненности вершин;
4. соотношение площади к периметру прямоугольников.

1.2. Алгоритмы расщепления вершин

Al-Badarneh's split [1] ищет две записи: с минимальной и максимальной суммой всех атрибутов. Эти записи считаются начальными и приписываются к соответствующим множествам. Оставшиеся записи сортируются по расстоянию между минимальным ключом первой записи и максимальным ключом текущей, первые m элементов результата добавляются в первое множество. Оставшиеся записи сортируются по расстоянию между максимальным ключом второй записи и минимальным ключом текущей. Теперь первые m элементов результата приписываются ко второму множеству. После этого каждая из оставшихся записей распределяется к ближайшей записи, используя те же расстояния, что применялись в предыдущих шагах. То есть оптимизируется четвертый критерий.

Ang-Tan linear split [2] разбивает записи вдоль какой-либо оси, в зависимости от того, к какой из сторон ограничивающего прямоугольника (относительно данной оси) эти записи больше прижаты. Ось же выбирается по следующим трем критериям:

1. разница количества записей, прижатых к противоположным сторонам ограничивающего прямоугольника должна быть минимальной;

2. если предыдущий критерий однозначно не определяет ось, выбирается та, для которой площадь перекрытия полученных прямоугольников минимальна;
3. если предыдущие два критерия также однозначно не определяют ось, выбирается та, для которой минимальна сумма площадей результирующих прямоугольников.

Оптимизируются второй и первый критерии соответственно.

Corner-based split [16] для каждого из углов ограничивающего прямоугольника считает количество записей, центры которых ближе всего расположены к этому углу. Далее происходит поиск плоскости разбиения, удовлетворяющей следующим условиям:

- Проходит через центр ограничивающего прямоугольника;
- Перпендикулярна одной из осей координат;
- Для искомой плоскости достигается наиболее равномерное распределение записей между углами, лежащими по разные стороны от этой плоскости.

Распределение по двум множествам происходит в зависимости от того, по какую сторону от полученной плоскости лежит центр записи. Как видно, здесь происходит оптимизация первого, второго и третьего критериев.

Double sort split [12] работает следующим образом: Пусть l_i, u_i — минимальный и максимальный атрибуты вдоль iой оси. Для каждой оси находятся пара $\langle a_i, b_i \rangle$, которая обладает следующими свойствами:

1. Все проекции разделяемых записей на iую ось лежат либо в промежутке (l_i, a_i) либо в (b_i, u_i);
2. В каждом из этих промежутков лежит по крайней мере m записей;
3. Для любого $a_i^{'} < a_i$ пара $\langle a_i^{'}, b_i \rangle$ не обладает первыми двумя свойствами;
4. Для любого $b_i^{'} > b_i$ пара $\langle a_i, b_i^{'} \rangle$ не обладает первыми двумя свойствами;
5. Среди всех возможных пар выбирается та, для которой $overlap = \frac{a_i - b_i}{u_i - l_i}$ минимален.

Далее выбирается та ось, для которой *overlap* соответствующей пары минимален. После этого, те пары, которые принадлежат только одному из промежутков (l_i, a_i), (b_i, u_i) распределяются по соответствующим множествам. Оставшиеся записи сортируются по разности расширения ограничивающих прямоугольников получившихся множеств. Затем k первых элементов приписываются первому множеству, а оставшиеся — второму, где k выбирается так, чтобы пересечение ограничивающих прямоугольников результирующих множеств было минимально. Таким образом, оптимизируются второй критерий.

Greene's split [7] находит две наиболее отдаленные друг от друга записи, и ось, вдоль которой нормированное расстояние между ними — максимально. После этого происходит сортировка записей по минимальному ключу в проекции на данную ось, и затем записи поровну распределяются между множествами. Если записей нечетное число, то средняя запись распределяется в то множество, которое она меньше расширит. Оптимизируются первый и третий критерии.

Guttman's linear split [8] сначала находит две записи, расстояние между которыми (нормированное по длине ограничивающего прямоугольника) вдоль какой-либо из осей — максимально. Далее каждая из оставшихся записей, приписывается к тому множеству, чей ограничивающей прямоугольник она расширит меньше всего. То есть оптимизируются первый и четвертый критерии.

Guttman's quadratic split [8] сначала находит две записи, поместив которые в одну вершину получили бы максимальную разницу между площадью их общего ограничивающего прямоугольника и суммой площадей их собственных ограничивающих прямоугольников. Эти записи выбираются как начальные. Далее последовательно выбирается та из оставшихся записей, для которой разница штрафа (увеличение площади расширяемого им ограничивающего прямоугольника) максимальна и эта запись приписывается к другому множеству. Здесь также оптимизируются первый и четвертый критерии.

K-means split [4] использует адаптацию популярного одноименного метода кластеризации при $K = 2$. В роли кластеризуемых объектов выступают ограничивающие прямоугольники дочерних записей.

1.3. R* и RR*-деревья

R*-дерево [15] — модификация R-дерева, с измененными алгоритмами выбора поддерева и расщепления вершин. Но, кроме того, существенной его особенностью является использование принципа принудительной повторной вставки записей (forced reinsert). Этот принцип основан на том факте, что структура R-дерева зависит от последовательности вставки в него новых записей и те из них, которые были вставлены на ранней стадии роста дерева, могут отрицательно влиять на качество структуры в целом. Поэтому используется механизм повторной вставки наиболее удаленных от центра ограничивающего прямоугольника записей переполненной вершины. Но реализация этого принципа довольно сложна в терминах многопоточной транзакционной системы, поэтому в данной работе будет рассматриваться его видоизмененный вариант.

RR*-дерево [3] — модификация R*-дерева, которая путем некоторых модификаций алгоритмов выбора поддерева и расщепления вершины.

Алгоритм выбора поддерева Для начала вычисляется множество записей, которые полностью покрывают вставляемую. Если оно не пусто, результатом является тот его элемент, объем (периметр) которого — минимален. Иначе, вычисляется увеличение периметра для каждой записи, как если бы новая запись была вставлена в данную. Все записи сортируются по данному значению. Результатом является первый элемент полученного набора, если периметр её пересечения с другими записями при этом не увеличится. В другом случае, алгоритм пытается добиться оптимального, относительно размера пересечения, выбора.

Алгоритм расщепления вершины Множество записей сортируется по проекциям минимального и максимального ключа на каждую из осей. Для каждого из наборов рассматривается $M+2-2m$ способов распределить вершины по множествам (первые i элементов в первое множество, а оставшиеся во второе). Таким образом, рассматривается $k(M+2-2m)$ вариантов разбить множество записей на два. Если разделяемая вершина — лист, рассматриваются только варианты, полученные с помощью проекции на ось, для которой сумма периметров

всех возможных конечных разбиений минимальна. Для каждого из оставшихся вариантов (вне зависимости от того, является ли данная вершина листом) вычисляется весовая функция: $w(i) = wg(i) * wf(i)$, где $wg(i)$ — весовая функция R*-дерева, $\frac{exp(-(\frac{x_i-\mu}{s(1+|\mu|)})^2)-exp(\frac{-1}{s^2})}{1-exp(\frac{-1}{s^2})}$, $\mu = (1 - \frac{2m}{M+1})\frac{\chi_c-\chi_o}{\lambda}$, s — константа, χ_c — текущий центр записи вдоль данной оси, χ_o — начальный центр записи вдоль данной оси, λ — длина записи вдоль данной оси. Искомый вариант разбиения получается посредством минимизации этой весовой функции.

1.4. Гильбертово R-дерево

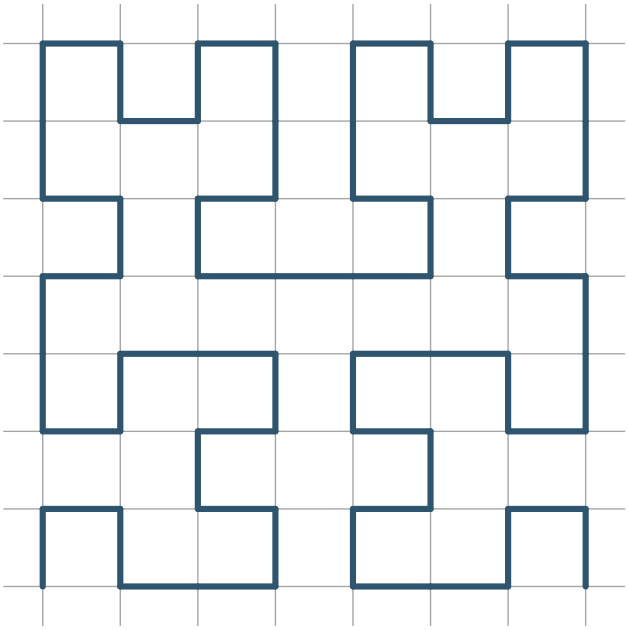

Рис. 2: Кривая Гильберта

Кривая Гильберта — одна из разновидностей кривых заполняющих всё пространство, т. е. эта кривая проходит все целочисленные точки k-мерного пространства ровно один раз и не пересекает себя. Таким образом, она позволяет однозначно занумеровать все целочисленные точки пространства. Пример кривой Гильберта 3-го порядка в двумерном пространстве приведен на Рис. 2. Было экспериментально показано [5], что кривая Гильберта позволяет достичь лучшей кластеризации при сравнении с некоторыми другими её аналогами. Основываясь на этих фактах, был предложен вариант R-дерева: Гильбертово R-дерево [10].

Его основная идея заключается в следующем:

- выполнять поисковые запросы так же как R-дерево;

- выполнять вставку записей, используя гильбертово значение (расстояние от начала координат до данной точки на Гильбертовой кривой) как ключ, и использовать политику отложенного расщепления вершины.

Это может быть достигнуто следующим образом: для каждой записи, помимо ограничивающего прямоугольника, можно хранить наибольшее гильбертово значение (НГЗ) для всех записей, лежащих в поддереве, задаваемом этой вершиной.

Рассмотрим подробнее специфику вставки записи в Гильбертово R-дерево. Для вставки нового элемента в вершине выбирается та запись, НГЗ которой минимально, но больше, чем гильбертово значение вставляемого элемента. В слчае же если такой записи нет, выбирается та НГЗ которой максимально.

Под политикой отложенного расщепления вершины, здесь понимается следующее. При вставке записи в переполненную вершину, если у нее есть не переполненные соседи, новая запись вставляется в одного из $s - 1$ соседей. Если же все $s - 1$ соседей данной вершины переполнены, создается новая. После чего все записи в данной вершине и $s - 1$ соседних с ней поровну перераспределяются, в зависимости от их НГЗ, по s вершинам в первом случае, или по $s + 1$ во втором.

2. GiST

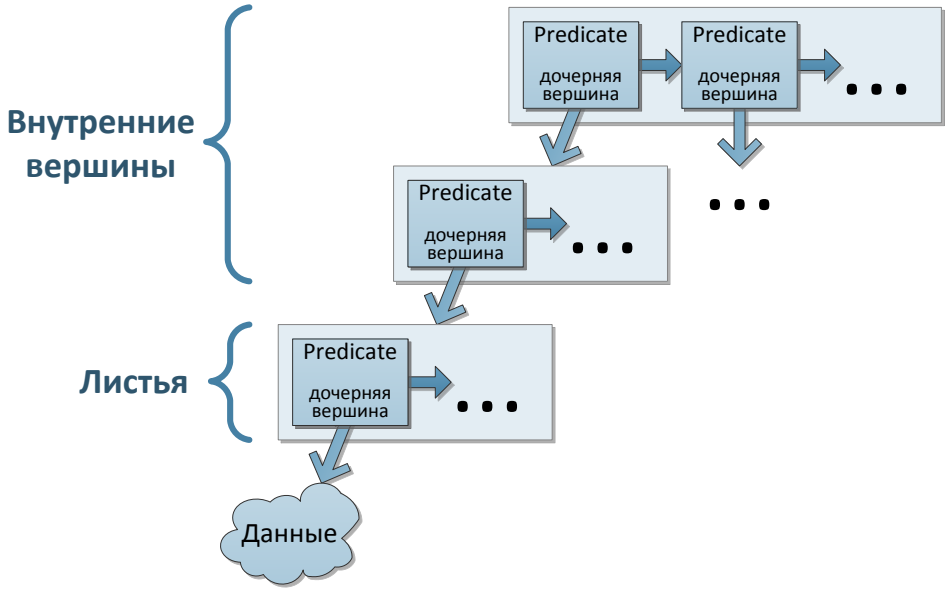

Рис. 3: GiST

GiST [9] — популярный шаблон, используемый для реализации поисковых деревьев. С его помощью можно реализовать B^+, R и многие другие деревья. Он представляет собой сбалансированное дерево поиска, где в каждой вершине может находиться от m до M элементов. Схематичное представление приведено на Рис. 3. В нем поддерживаются операции вставки, поиска, обновления, и удаления записей. Их реализация использует реализуемые разработчиком структуру `Predicate` и шесть методов описанных ниже.

- `Consistent(`E, q`)`: По записи $E = (p, ptr)$, и предикату q, возвращает $false$, если $p \wedge q$ гарантированно не выполняется, и $true$ в противном случае.
- `Union(`P`)`: По множеству записей P, возвращает предикат r, который принимает истинное значение для каждой из этих записей.
- `Compress(`E`)`: По записи $E = (p, ptr)$, возвращает такую запись (p', ptr'), что p' — сжатое представление p.
- `Decompress(`E`)`: По сжатому представлению записи $E = (p', ptr)$, где $p' =$ `Compress(`p`)`, возвращает такую запись (r, ptr), что $p \to r$.

- `Penalty(`E_1, E_2`)`: По двум записям $E_1 = (p_1, ptr_1)$, $E_2 = (p_2, ptr_2)$, возвращает штраф за вставку записи E_2 в поддерево с корнем в E_1.
- `PickSplit(`P`)`: Разделяет множество записей P на два множества: P_1, P_2, каждое из которых содержит, по крайней мере, по kM элементов.

Реализация дерева по шаблону GiST обладает следующими преимуществами:

- легкость реализации новой структуры (необходимо реализовать всего 6 методов интерфейса GiST);
- инкапсуляция внутренней структуры дерева;
- GiST располагает механизмом обеспечения параллельного доступа [11];
- GiST располагает механизмом обеспечения изоляции транзакций [11].

Основным же её недостатком является то, что не любую структуру данных можно реализовать таким способом (R^+-дерево, B-дерево, R*-дерево, и др.). Например R^+-дерево не может быть встроено по той причине, что оно позволяет хранить одну и ту же запись в нескольких листьях.

3. Анализ возможности встраивания дополнительных структур данных в GiST

Все рассмотренные алгоритмы расщепления вершин без всяких проблем могут быть выражены в терминах метода GiST `PickSplit`, так как они оперируют только характеристиками разделяемых записей: размер, положение в пространстве. Однако рассмотренные более глубокие модификации R-дерева, а именно RR*-дерево и Гильбертово R-дерево, создают некоторые трудности для встраивания их в GiST.

3.1. Встраивание RR*-дерева в GiST

При решении задачи встраивания RR*-дерева в GiST на этапе анализа были выделены следующие проблемы.

- `ChooseSubTree` метод не может быть выражен через метод GiST `Penalty` потому, что ему необходимо иметь доступ ко всем записям обрабатываемой вершины. Он использует адаптивную стратегию, основанную на использовании объема или периметра прямоугольников.
- `Split` метод не может быть выражен через метод GiST `PickSplit` по той причине, что он должен иметь доступ не только к разделяемым записям, но и к информации об истории изменения данной вершины. А именно, ему необходимо знать, где находился центр вершины на момент её создания и является ли эта вершина листом.

3.2. Встраивание Гильбертова R-дерева в GiST

При анализе Гильбертова R-дерева были выявлены следующие трудности.

- В архитектуре GiST не предусмотрена возможность использования механизма отложенного расщепления. Здесь расщепление происходит каждый раз при попытке вставить запись в переполненную вершину. Однако это является частным случаем данной политики при $s = 1$. Поэтому в оригинальный GiST можно вложить только Гильбертово R-дерево с такой политикой расщепления.

- Кроме того, возникают проблем при работе с НГЗ. Возможно хранить НГЗ в поле структуры `Predicate`, и реализовать метод `Union` так, чтобы все записи находились в консистентном состоянии. Тогда метод `PickSplit` может быть легко реализован, путем разбиения набора сортированных по НГЗ записей на две примерно равные части. Однако эффективно реализовать алгоритм выбора поддерева через метод `Penalty` не так просто. Возможно, например, реализовать его следующим образом:

Вход: a_i - обрабатываемая запись, x - вставляемая запись

1: **if** $a_i.lhv \geq x.lhv$ **then** {lhv - наиболшее гильбертово значение}
2: **return** $a_i.lhv - x.lhv$
3: **else**
4: **return** $x.lhv - a_i.lhv + N$ {Здесь N - верхний порог гильбертового значения в данном контексте}
5: **end if**

Но эта реализация неудачна по следующим причинам:

  - она неэффективна в том плане, что проводится довольно много операций с гильбертовыми значениями;
  - необходимо заранее знать домены всех атрибутов для указания верхнего порога гильбертовых значений N.

4. Модификация структуры данных GiST

Для решения описанных выше проблем предлагается следующая модификация GiST.

- Добавить структуру: `NodeInfo`
- Добавить новый пользовательский метод: `UpdateNodeInfo(Node node, NodeInfo info);`
- Заменить метод `Penalty(Entry` e_1`, Entry` e_2`);` на: `Entry FindOptimalEntry(Ent entries, entry_to_insert);`
- Изменить сигнатуру метода `PickSplit` с: `PickSplit(Entry[] entries);` на: `PickSplit(Entry[] entries, NodeInfo info);`

Рассмотрим подробнее предложенные модификации:

Поскольку некоторые из вышеописанных вариантов R-дерева на этапе выбора дочернего узла требуют доступа ко всем дочерним узлам, предлагается выделить эту функциональность в отдельный пользовательский метод `FindOptimalEntry`. Этот метод по списку записей должен выбрать ту, в которую оптимальнее всего вставить новую. При этом, те структуры данных, которым не требуются такая функциональность, могут продолжать использовать метод `Penalty`.

Метод `UpdateNodeInfo` используется для сохранения некоторой пользовательской информации в структуру `NodeInfo`, ассоциированную с текущим узлом. Например, в RR*-дереве для проведения операции расщепления вершины необходима информация о центре вершины в момент её создания и о том, является ли эта вершина листом. В случае необходимости, пользователь может предоставить реализацию структуры `NodeInfo`.

Для поддержки представленных модификаций, необходимо произвести следующие изменения в ключевых методах GiST:

- метод `LocateLeaf` вместо `Penalty` теперь вызывает метод `FindOptimalEntry`;
- метод `Split` изменен следующим образом:

  - `PickSplit` вызывается с дополнительным аргументом `NodeInfo`, который однозначно определяется расщепляемой вершиной (например, как её поле);
  - `UpdateNodeInfo` вызывается для обеих вершин, полученных в результате расщепления;

- `UpdateNodeInfo` вызывается для корня при вставке самой первой записи в дерево.

Стоит отметить, что метод `UpdateNodeInfo` может, при необходимости, вызываться и в других ситуациях (например, если алгоритму расщепления необходима подробная история изменений вершины).

Предложенная модификация не влияет на механизм поддержки многопоточного доступа, так как все алгоритмы модификации работают с элементами вершины, которая на момент вызова соответствующих методов заблокирована согласно алгоритмам GiST.

- Метод `LocateLeaf` перед поиском подходящей записи, среди дочерних для данной вершины, берет, в зависимости от того, является ли текущая вершина листом, разделяемую или неразделяемую блокировку на эту вершину. Каждого из этих вариантов достаточно, так как метод `FindOptimalEntry` не изменяет состояние дерева.

- Метод `PickSplit` начинается с обработки вершины, на которую взята неразделяемая блокировка. Он также берет неразделяемые блокировки на родительскую вершину и на новую созданную. Поэтому если метод `UpdateNodeInfo` будет вызван до снятия этих блокировок, механизм обеспечения многопоточного доступа не пострадает.

- При вставке самой первой вершины, корень является листом, поэтому на него берется неразделяемая блокировка, значит можно без опасений вызвать для корня метод `UpdateNodeInfo`.

4.1. Модификация прототипа

Описаная модификация GiST была реализована в прототипе системы многомерного индексирования, разрабатываемой на кафедре системного программирования математико-механического факультета СПбГУ.

С помощью нового шаблона GiST, в систему было встроено RR*-дерево. Метод `FindOptimalEntry` был реализован как "CSRevised'' алгоритм, описанный в [3]. Метод `PickSplit` был реализован как "Split'' алгоритм, описаный в [3]. Были необходимы лишь поверхностные модификации данных алгоритмов для адаптации к интерфейсу GiST. Также структура `NodeInfo`, была определена следующим образом:

```
struct NodeInfo {
        Predicate* bounding_box_original_center;
        bool is_leaf;
}
```

Наконец, `UpdateNodeInfo` сохраняет центр текущего ограничивающего прямоугольника в поле `bounding_box_original_center` и выставляет нужное значение для поля `is_leaf`.

Также описанные модификации позволяют эффективно реализовать алгоритм выбора поддерева в Гильбертовом R-дереве.

5. Эксперименты

Для проведения экспериментального сравнения алгоритмов расщепления вершин в R-дереве, был использован упомянутый ранее прототип системы многомерного индексирования. Он содержит все необходимые компоненты для проведения сравнения, и в нем были реализованы все изученные алгоритмы.

5.1. Описание тестового стенда

В эксперимантах был использован тестовый стенд со следующими характеристиками:

- Аппаратное обеспечение
  - Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
  - 8GB RAM
- Программное обеспечение
  - x86_64 GNU/Linux, ядро 3.8.0-26-generic

5.2. Проведение измерений

Была проведена серия экспериментов для изучения производительности различных вариаций R-дерева. Рассматривались два критерия качества изучаемой структуры:

Время построения — общее время, затраченное на заполнения индекса фиксированным набором начальных данных;

Пропускная способность — количество транзакций за единицу времени, исполненных с использованием построенного индекса.

В качестве начальных данных использовались записи общим объемом 32 Mb, где ключами были точки пространства размерности $2, 4, 6, 8$, удовлетворяющие следующим распределениям:

- нормальное;

- равномерное;
- Парето.

Среди наборов запросов к индексу рассматривались варианты, в которых преобладают либо запросы на диапазон (Range query) с селективностью 0.001, либо точечные запросы (Point query).

Для каждой конфигурации были проведены измерения производительности при исполнении запросов $1, 2, 4$ потоками.

5.3. Результаты измерений

Результаты экспериментов представлены в виде столбиковых диаграмм в Приложении А. Время построения индекса и пропускная способность оценивались при помощи 95% доверительных интервалов.

Эксперименты показали следующие результаты.

- На этапе построения индекса худший результат практически по всем типам нагрузки показывает RR*-дерево, однако при малой размерности пространства и большом количестве потоков разница становится несущественной. После него следуют Гильбертово R-дерево и R-дерево с Guttman's quadratic split. Остальные алгоритмы демонстрирует одинаково хорошую производительность.
- На этапе исполнения транзакций алгоритмы показали себя по разному на данных большой размерности в зависимости от типов запросов. На данных малой размерности во всех случаях лидирует RR*-дерево, но его производительность в большинстве ситуаций сильно падает с увеличением размерности пространства.
  - При преобладании запросов на диапазон на большей части нагрузок хорошо себя показало Гильбертово R-дерево. Стоит отметить, что в данной ситуации его отставание от RR*-дерева на данных малой размерности практически незначимо. Также в некоторых ситуациях хорошую пропускную способность показывают R-дерево с K-means split, R-дерево с Guttman's quadratic split.

- В случае преобладания точечных запросов были получены разные результаты в зависимости от распределения начальных данных.
  * При равномерно распределенных данных лучшую производительность показало R-дерево с Greene's split. Несколько хуже себя показало Гильбертово R-дерево.
  * При нормально распределенных данных картина обратная: при увеличении размерности пространства вперед выходит Гильбертово R-дерево, за ним же следует R-дерево с Greene's split.
  * Для распределения Парето наблюдается гораздо более медленное снижение производительности RR*-дерева при увеличении размерности пространства. В связи с этим RR*-дерево лидирует по результатам большей части экспериментов. Однако в пространстве размерности 8 R-дерево с Greene's split значительно его опережает.

Заключение

В рамках дипломной работы были выполнены следующие задачи:

1. проведен обзор десяти основных существующих подходов к решению задачи расщепления вершин в R-дереве;
2. проведен анализ возможности встраивания изученных модификаций R-дерева в структуру данных GiST, по результатам которого было принято решение модифицировать последнюю;
3. проведена модификация структуры данных GiST для поддержки дополнительных деревьев поиска;
4. проведено экспериментальное сравнение алгоритмов пространственного индексирования, которое показало, что оптимальным выбором в большинстве случаев будет Гильбертово R-дерево, однако при некоторых условиях хорошие результаты показывают RR*-дерево, R-дерево с Greene's split, R-дерево с K-means split, R-дерево с Guttman's quadratic split.

По теме работы была опубликована статья на конференции СПИСОК'14: "Supporting additional tree data structures in GiST'' Pavel Fedotovsky, Kirill Cherednik and Chernishev George.

Список литературы

[1] Al-Badarneh Amer F., Yaseen Qussai, Hmeidi Ismail. A New Enhancement to the R-tree Node Splitting // J. Inf. Sci. — 2010. — feb. — Vol. 36, no. 1. — P. 3–18. — URL: http://dx.doi.org/10.1177/0165551509340360.

[2] Ang C.H., Tan T.C. New linear node splitting algorithm for R-trees // Advances in Spatial Databases / Ed. by Michel Scholl, Agnès Voisard. — Springer Berlin Heidelberg, 1997. — Vol. 1262 of Lecture Notes in Computer Science. — P. 337–349. — URL: http://dx.doi.org/10.1007/3-540-63238-7_38.

[3] Beckmann Norbert, Seeger Bernhard. A Revised R*-tree in Comparison with Related Index Structures // Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data. — SIGMOD '09. — New York, NY, USA : ACM, 2009. — P. 799–812. — URL: http://doi.acm.org/10.1145/1559845.1559929.

[4] Brakatsoulas Sotiris, Pfoser Dieter, Theodoridis Yannis. Revisiting R-Tree Construction Principles // Proceedings of the 6th East European Conference on Advances in Databases and Information Systems. — ADBIS '02. — London, UK, UK : Springer-Verlag, 2002. — P. 149–162. — URL: http://dl.acm.org/citation.cfm?id=646046.676614.

[5] Faloutsos C., Roseman S. Fractals for Secondary Key Retrieval // Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. — PODS '89. — New York, NY, USA : ACM, 1989. — P. 247–252. — URL: http://doi.acm.org/10.1145/73721.73746.

[6] Gaede Volker, Günther Oliver. Multidimensional Access Methods // ACM Comput. Surv. — 1998. — jun. — Vol. 30, no. 2. — P. 170–231. — URL: http://doi.acm.org/10.1145/280277.280279.

[7] Greene D. An implementation and performance analysis of spatial data access methods // Data Engineering, 1989. Proceedings. Fifth International Conference on. — 1989. — feb. — P. 606–615.

[8] Guttman Antonin. R-trees: A Dynamic Index Structure for Spatial Searching // Proceedings of the 1984 ACM SIGMOD International Conference on Management

of Data. — SIGMOD '84. — New York, NY, USA : ACM, 1984. — P. 47–57. — URL: http://doi.acm.org/10.1145/602259.602266.

[9] Hellerstein Joseph M., Naughton Jeffrey F., Pfeffer Avi. Generalized Search Trees for Database Systems // Proceedings of the 21th International Conference on Very Large Data Bases. — VLDB '95. — San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1995. — P. 562–573. — URL: http://dl.acm.org/citation.cfm?id=645921.673145.

[10] Kamel Ibrahim, Faloutsos Christos. Hilbert R-tree: An Improved R-tree Using Fractals // Proceedings of the 20th International Conference on Very Large Data Bases. — VLDB '94. — San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1994. — P. 500–509. — URL: http://dl.acm.org/citation.cfm?id=645920.673001.

[11] Kornacker Marcel, Mohan C., Hellerstein Joseph M. Concurrency and Recovery in Generalized Search Trees // Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data. — SIGMOD '97. — New York, NY, USA : ACM, 1997. — P. 62–72. — URL: http://doi.acm.org/10.1145/253260.253272.

[12] Korotkov A. A New Double Sorting-based Node Splitting Algorithm for R-tree // Program. Comput. Softw. — 2012. — jun. — Vol. 38, no. 3. — P. 109–118. — URL: http://dx.doi.org/10.1134/S0361768812030024.

[13] Liu Ling, Zsu M. Tamer. Encyclopedia of Database Systems. — 1st edition. — Springer Publishing Company, Incorporated, 2009. — ISBN: 0387355448, 9780387355443.

[14] R-Tree (and Family) / Apostolos N. Papadopoulos, Antonio Corral, Alexandros Nanopoulos, Yannis Theodoridis // Encyclopedia of Database Systems. — 2009. — P. 2453–2459.

[15] The R*-tree: An Efficient and Robust Access Method for Points and Rectangles / Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, Bernhard Seeger // Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data. — SIGMOD '90. — New York, NY, USA : ACM, 1990. — P. 322–331. — URL: http://doi.acm.org/10.1145/93597.98741.

[16] Sleit Azzam, Al-Nsour Esam. Corner-based splitting: An improved node splitting algorithm for R-tree // Journal of Information Science. — 2014. — Vol. 40, no. 2. — P. 222–236. — http://jis.sagepub.com/content/40/2/222.full.pdf+html.

A. Результаты экспериментов

В этом приложении представлены результаты экспериментов в виде столбиковых диаграмм.

Для сравниваемых алгоритмов и варьируемых параметров нагрузочных данных использовались следующие обозначения.

- Алгоритмы:

  Hilbert — Гильбертово R-дерево;

  RR* — RR*-дерево;

  AB — R-дерево с Al-Badarneh's split;

  AT — R-дерево с Ang-Tan linear split;

  CB — R-дерево с Corner-based split;

  DS — R-дерево с Double sort split;

  GR — R-дерево с Greene's split;

  GL — R-дерево с Guttman's linear split;

  GQ — R-дерево с Guttman's quadratic split;

  KM — R-дерево с K-means split.

- Функции распределения:

  Равном. — Равномерное распределение;

  Норм. — Нормальное распределение;

  Парето — Распределение Парето.

- Типы запросов:

  point — Точечные запросы (point query);

  range — Запросы на диапазон (range query).

- Размерность пространства обозначается как "dim - N".

A.1. Время построения

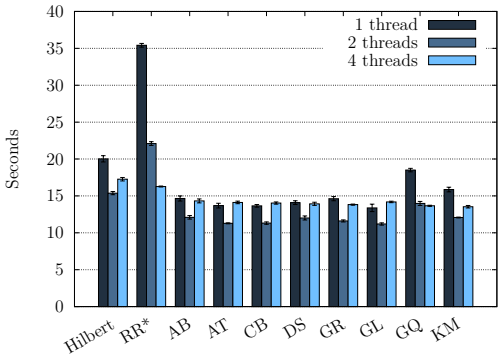

Рис. 4: Равном., dim - 2

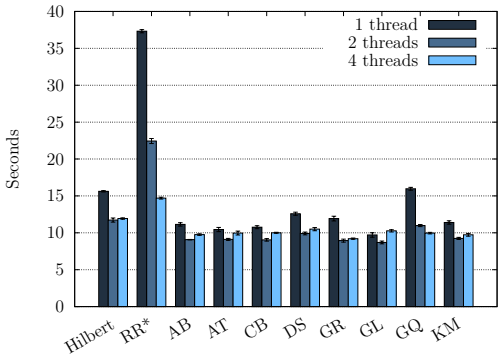

Рис. 5: Равном., dim - 4

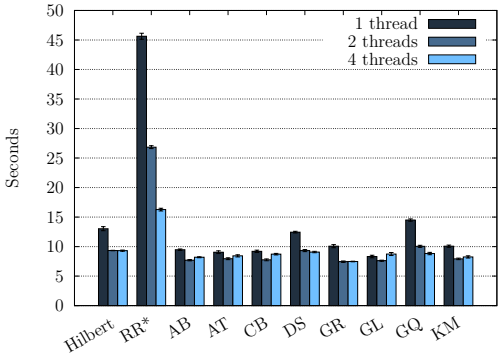

Рис. 6: Равном., dim - 6

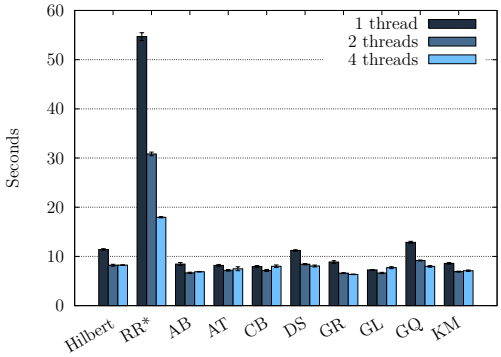

Рис. 7: Равном., dim - 8

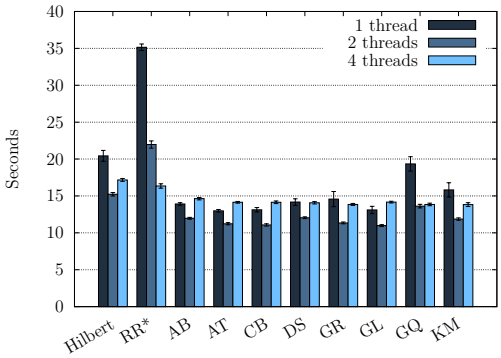

Рис. 8: Норм., dim - 2

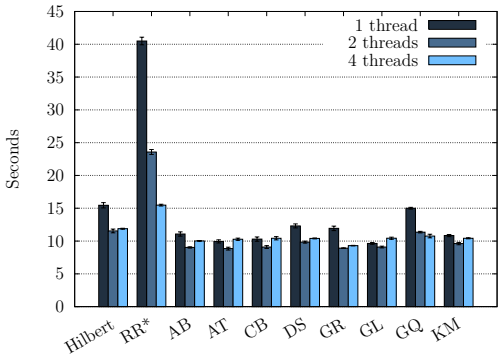

Рис. 9: Норм., dim - 4

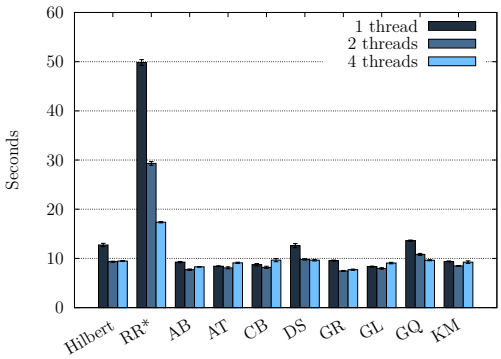

Рис. 10: Норм., dim - 6

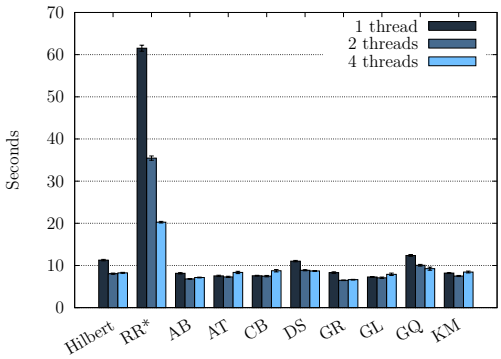

Рис. 11: Норм., dim - 8

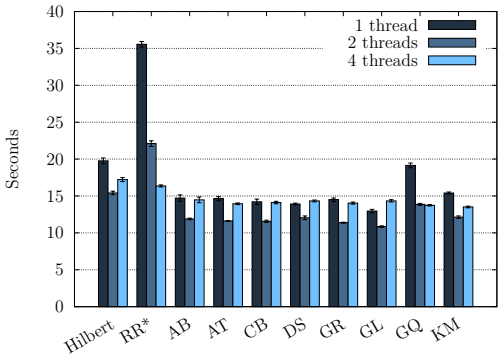

Рис. 12: Парето, dim - 2

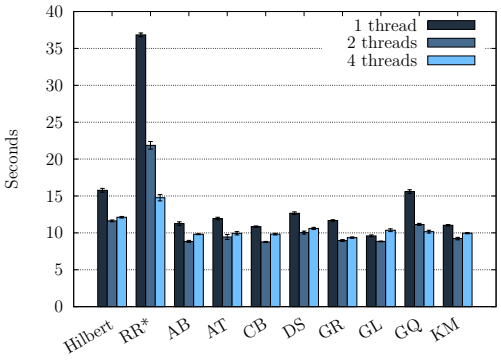

Рис. 13: Парето, dim - 4

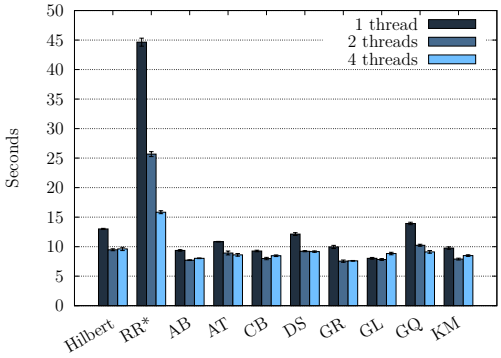

Рис. 14: Парето, dim - 6

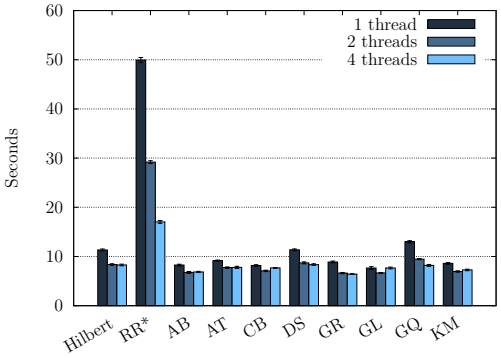

Рис. 15: Парето, dim - 8

A.2. Пропускная способность

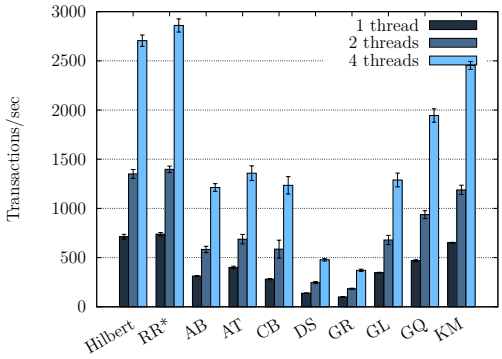

Рис. 16: Равном., dim - 2, range

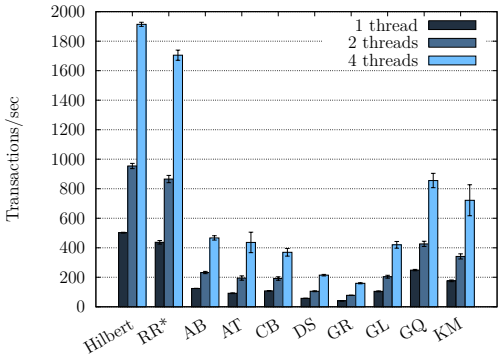

Рис. 17: Равном., dim - 4, range

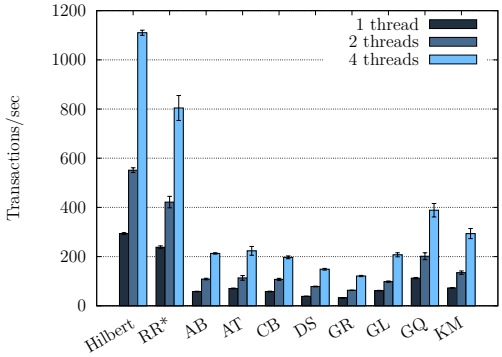

Рис. 18: Равном., dim - 6, range

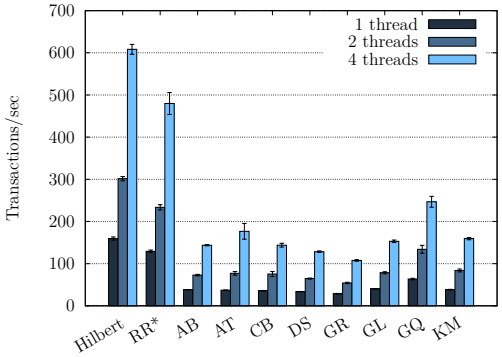

Рис. 19: Равном., dim - 8, range

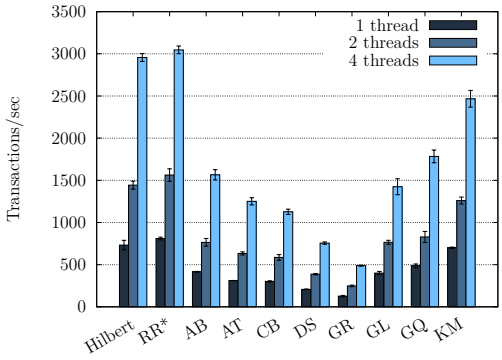

Рис. 20: Норм., dim - 2, range

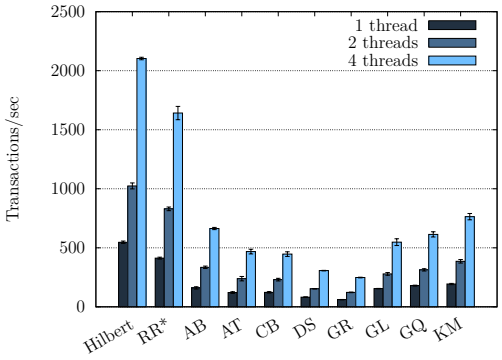

Рис. 21: Норм., dim - 4, range

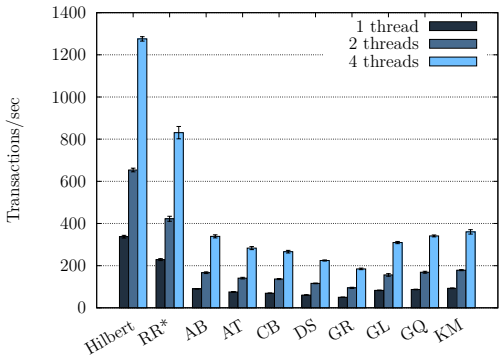

Рис. 22: Норм., dim - 6, range

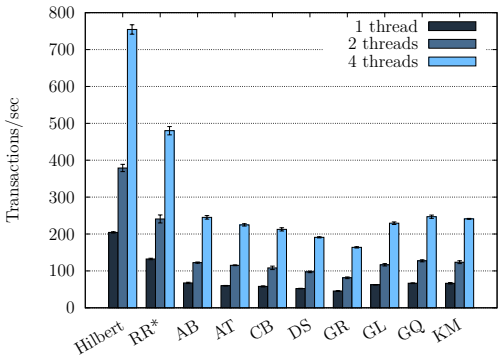

Рис. 23: Норм., dim - 8, range

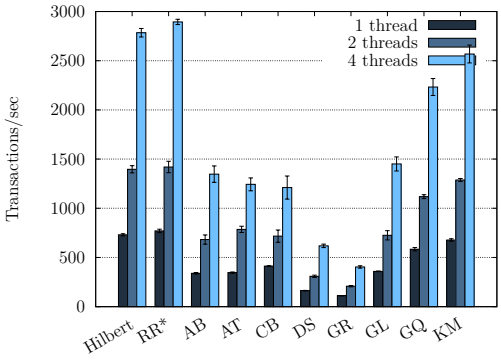

Рис. 24: Парето, dim - 2, range

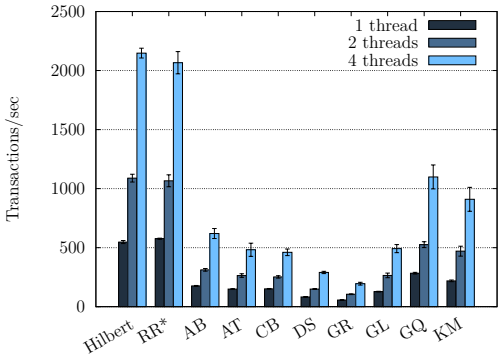

Рис. 25: Парето, dim - 4, range

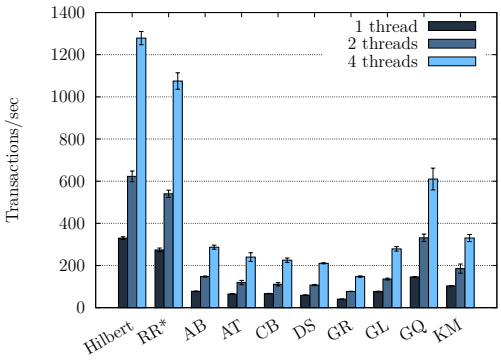

Рис. 26: Парето, dim - 6, range

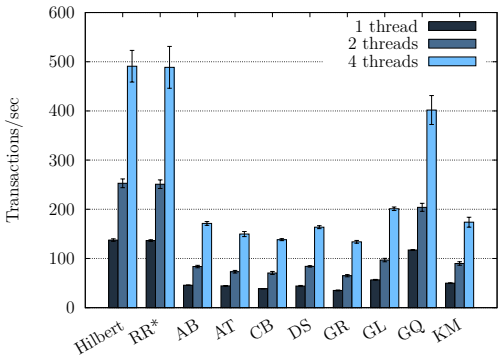

Рис. 27: Парето, dim - 8, range

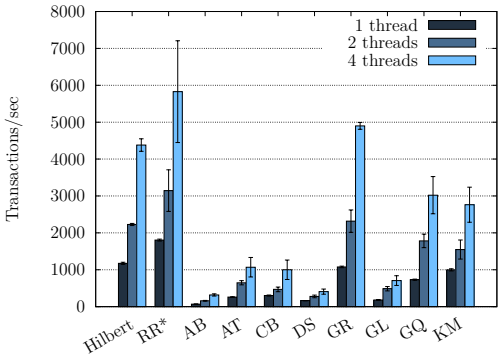

Рис. 28: Равном., dim - 2, point

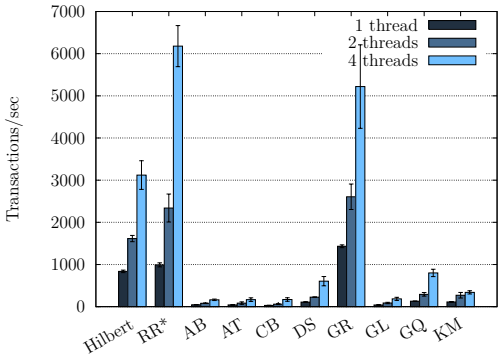

Рис. 29: Равном., dim - 4, point

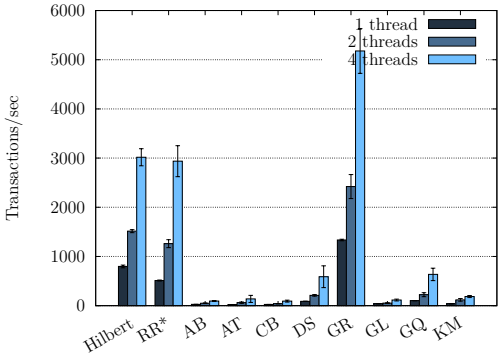

Рис. 30: Равном., dim - 6, point

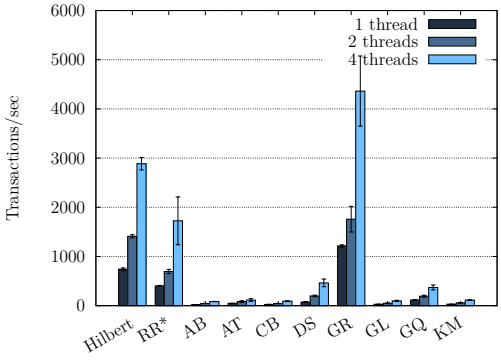

Рис. 31: Равном., dim - 8, point

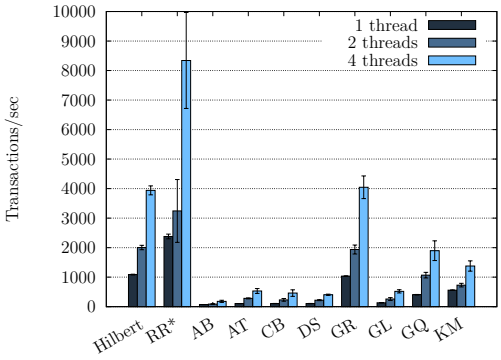

Рис. 32: Норм., dim - 2, point

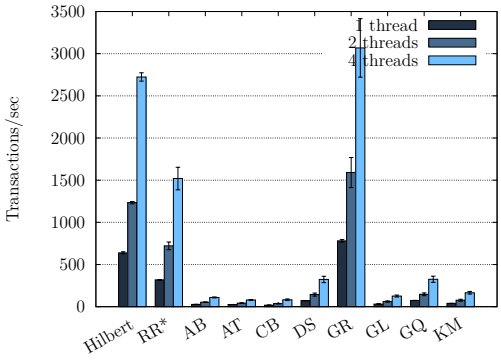

Рис. 33: Норм., dim - 4, point

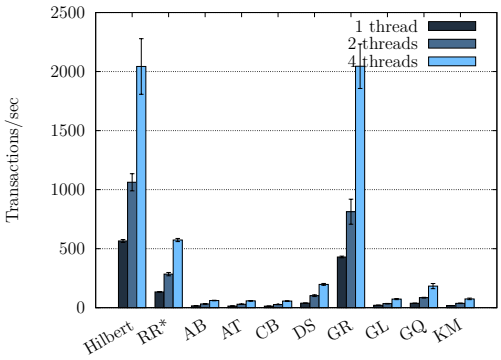

Рис. 34: Норм., dim - 6, point

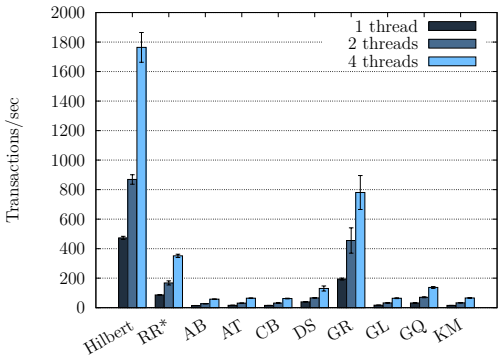

Рис. 35: Норм., dim - 8, point

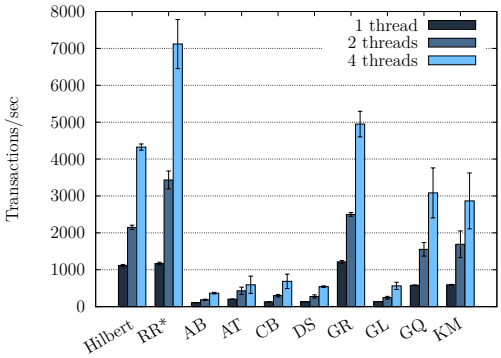

Рис. 36: Парето, dim - 2, point

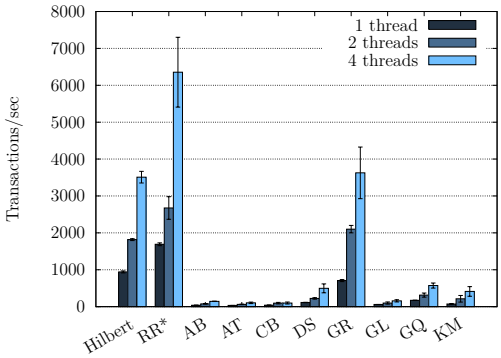

Рис. 37: Парето, dim - 4, point

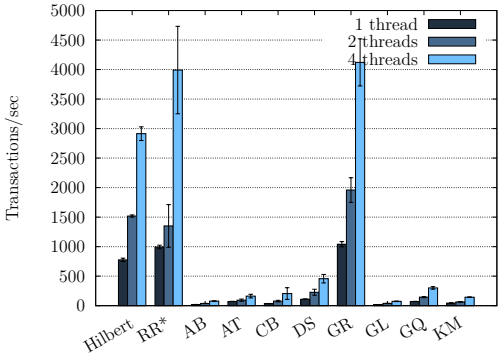

Рис. 38: Парето, dim - 6, point

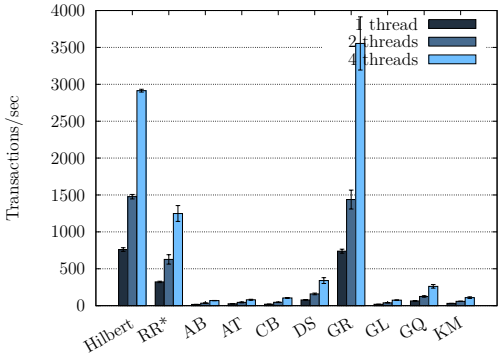

Рис. 39: Парето, dim - 8, point