Эффективный поиск ближайших соседей в PostgreSQL

Oleg Bartunov, Teodor Sigaev
Sternberg Astronomical Institute,
Moscow University
Knn-search: The problem

- What are interesting points near Royal Oak pub in Ottawa?
- What are the closest events to the May 20, 2009 in Ottawa?
- Similar images – feature extraction, Hamming distance
- Classification problem (major voting)
- ............
- GIS, Science (high-dimensional data)
- Data merging (de-duplication)
Knn-search: Existing solutions

```
knns# select id, date, event from events order by date <-> '1957-10-04':::date asc limit 10;

<table>
<thead>
<tr>
<th>id</th>
<th>date</th>
<th>event</th>
</tr>
</thead>
<tbody>
<tr>
<td>58137</td>
<td>1957-10-04</td>
<td>U.S.S.R. launches Sputnik I, 1st artificial Earth satellite</td>
</tr>
<tr>
<td>58136</td>
<td>1957-10-04</td>
<td>&quot;Leave It to Beaver,&quot; debuts on CBS</td>
</tr>
<tr>
<td>117062</td>
<td>1957-10-04</td>
<td>Gregory T Linteris, Demarest, New Jersey, astronaut, sk: STS 83</td>
</tr>
<tr>
<td>117061</td>
<td>1957-10-04</td>
<td>Christina Smith, born in Miami, Florida, playmate, Mar, 1978</td>
</tr>
<tr>
<td>102670</td>
<td>1957-10-05</td>
<td>Larry Saumell, jockey</td>
</tr>
<tr>
<td>31456</td>
<td>1957-10-03</td>
<td>Willy Brandt elected mayor of West Berlin</td>
</tr>
<tr>
<td>58291</td>
<td>1957-10-05</td>
<td>12th Ryder Cup: Britain-Ireland, 7 -4 at Lindrick GC, England</td>
</tr>
<tr>
<td>58290</td>
<td>1957-10-05</td>
<td>11th NHL All-Star Game: All-Stars beat Montreal 5-3 at Montreal</td>
</tr>
<tr>
<td>58292</td>
<td>1957-10-05</td>
<td>Yugoslav dissident Milovan Djilos sentenced to 7 years</td>
</tr>
<tr>
<td>102669</td>
<td>1957-10-05</td>
<td>Jeanne Evert, tennis player, Chris' sister</td>
</tr>
</tbody>
</table>
```

Time: 115.548 ms

- **Very inefficient:**
  - Full table scan, btree index on date won't help.
  - Sort full table
• Find 10 closest points in table T
  MS SQL the best solution: Number is a table with integers

DECLARE @start FLOAT = 1000;
WITH NearestPoints AS
(
    SELECT TOP(1) WITH TIES *, T.g.STDistance(@x) AS dist
    FROM Numbers JOIN T WITH(INDEX(spatial_index))
    ON T.g.STDistance(@x) < @start*POWER(2,Numbers.n)
    ORDER BY n
)
SELECT TOP(10) * FROM NearestPoints
ORDER BY n, dist

Denali supports knn, but still lose the above trick :)
Knn-search: Existing solutions

- Oracle Spatial has SDO_NN function (only 2 dimensions!)
  http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14255/sdo_operat.htm#i78067

SELECT /*+ INDEX(c cola_spatial_idx) */
c.mkt_id, c.name FROM cola_markets c WHERE SDO_NN(c.shape,
sdo_geometry(2001, NULL, sdo_point_type(10,7,NULL), NULL,
NULL), 'sdo_num_res=2') = 'TRUE';
Knn-search: Existing solutions

- Traditional way to speedup query
  - Use indexes - very inefficient (no search query !)
    - Scan full index
    - Full table scan, but in random order !
    - Sort full table
    - Better not to use index at all !
  - Constrain data space (range search)
    - Incremental search → too many queries
    - Need to know in advance size of neighbourhood, how ?
      1Km is ok for Paris, but too small for Siberia
    - Maintain 'density map' ?
What's a neighbourhood?
Knn-search: What do we want!

- We want to avoid full table scan – read only <right> tuples
  - So, we need index
- We want to avoid sorting – read <right> tuples in <right> order
  - So, we need special strategy to traverse index
- We want to support tuples visibility
  - So, we should be able to resume index traverse
R-tree index
• Visualization of R-tree index using Gevel.

• Greece
  (data from rtreeportal.org)
SELECT *
FROM events
WHERE events.coord <@ 'QUERY';

• Root page: R1, R2 keys
• Inner pages: I3, I2 keys
• Leaf pages: 4 points

• Very efficient for Search!
Knn-search: Index traverse

- Depth First Search (stack, LIFO)
  R-tree search

- Breadth First Search (queue, FIFO)

- Both strategies are not good for us – full index scan
Knn-search: Index traverse

- **Best First Search** (PQ, priority queue). Maintain order of items in PQ according their distance from given point
  - Distance to MBR (rectangle for Rtree) for internal pages – minimum distance of all items in that MBR
  - Distance = 0 for MBR with given point
  - Distance to point for leaf pages
- Each time we extract point from PQ we output it – it is next closest point! If we extract rectangle, we expand it by pushing their children (rectangles and points) into the queue.
- We traverse index by visiting only interesting nodes!
• Simple example – non-overlapped partitioning
Knn-search: Index traverse

- Simple example – non-overlapped partitioning

- Priority Queue
  - 1: \{1,2,3,4,5,6,7,8,9\}
  - 2: \{2,5,6,7,9\}, \{1,3,4,8\}
  - 3: \{5,6,7,9\}, \{1,3,4,8\}, \{2\}
  - 4: \{5,9\}, \{1,3,4,8\}, \{2\}, \{6,7\}
  - 5: \{1,3,4,8\}, 5, \{2\}, \{6,7\}, 9
  - 6: \{1,3,4\}, \{8\}, 5, \{2\}, \{6,7\}, 9
  - 7: 4, \{8\}, 5, \{2\}, \{6,7\}, 3, 1, 9

  *we can output 4 without visit other rectangles!*

  - 8: 5, \{2\}, \{6,7\}, 3, 8, 1, 9
  - 9: \{6,7\}, 3, 2, 8, 1, 9
  - 10: 3, 2, 8, 1, 9, 6, 7
Knn-search: Index traverse

- Simple example – non-overlapped partitioning

- Priority Queue
  - 1: \{1, 2, 3, 4, 5, 6, 7, 8, 9\}
  - 2: \{2, 5, 6, 7, 9\}, \{1, 3, 4, 8\}
  - 3: \{5, 6, 7, 9\}, \{1, 3, 4, 8\}, \{2\}
  - 4: \{5, 9\}, \{1, 3, 4, 8\}, \{2\}, \{6, 7\}
  - 5: \{1, 3, 4, 8\}, 5, \{2\}, \{6, 7\}, 9
  - 6: \{1, 3, 4\}, \{8\}, 5, \{2\}, \{6, 7\}, 9
  - 7: \{4\}, \{8\}, 5, \{2\}, \{6, 7\}, 3, 1, 9
  - 8: \{5\}, \{2\}, \{6, 7\}, 3, 8, 1, 9
Knn-search: Performance

- **SEQ (no index)** – base performance
  - Sequentially read full table + Sort full table (can be very bad, sort_mem !)

- **DFS** – very bad !
  - Full index scan + Random read full table + Sort full table

- **BFS** – the best for small k !
  - Partial index scan + Random read k-records
    - \( T(\text{index scan}) \sim \text{Height of Search tree} \sim \log(n) \)
  - Performance win BFS/SEQ \( \sim \text{Nreipages/k}, \text{for small k. The more rows, the more benefit} \)!
  - Can still win even for \( k=n \) (for large tables) - no sort !
Knn-search: What do we want!

- We want to avoid full table scan – read only <right> tuples
  - So, we need index
- We want to avoid sorting – read <right> tuples in <right> order
  - So, we need special strategy to traverse index
- We want to support tuples visibility
  - So, we should be able to resume index traverse
- We want to support many data types
  - So, we need to modify GiST
Knn-search: modify GiST

• GiST – Generalized Search Tree, provides
  • API to build custom disk-based search trees (any tree, where key of internal page is a Union of keys on children pages)
  • Recovery and Concurrency
  • Data type and query extendability

• GiST is widely used in GIS (PostGIS), text search,…

• Current strategy of search tree traverse is DFS
  • Not good for knn-search
  • We need to add Best First Search strategy for knn-search
  • Retain API compatibility
- Knn-query uses ORDER BY clause

```sql
SELECT ... FROM ... WHERE ...
ORDER BY p < - > '(0.,0.)'::point
LIMIT k;
```

< - > - distance operator, should be provided for data type
- compress/decompress
- same
- union
- penalty
- picksplit
- consistent
- **distance** - is needed if the operator class wishes to support ordered scans (nearest-neighbor searches)
Depth First Search

Push BlkNo of root into stack

Is stack empty?

Y

Go away

N

Push all matched BlkNos into stack

Is a leaf?

Y

Return all matched entries

N

Pop BlkNo and read page
KNN-search: Priority Queue

Push (BlkNo of root, distance = 0)

Is queue empty?

- Y: Go away
- N: Pop (pointer, distance)

Is a pointer to heap?

- Y: Return pointer
- N: Read index page, push matched pairs (pointer, distance) Distance is a result of consistent methods
GiST: Technical details

- Priority queue implemented as a RB-tree (Red-Black tree)
- Each node of RB-tree contains a list of pointers - pointers to internal pages follow pointers to heap.
Depth First Search

```c
push Stack, Root;
While Stack {
    If p is heap {
        output p;
    } else {
        children = get_children(p);
        push Stack, children;
    }
}
```

Best First Search

```c
push PQ, Root;
While PQ {
    If p is heap {
        output p;
    } else {
        Children = get_children(p);
        push PQ, children;
    }
}
```

- For non-knn search all distances are zero, so PQ => Stack and BFS => DFS
- We can use only one strategy for both – normal search and knn-search!
Knn-search: What do we want!

- We want to avoid full table scan – read only <right> tuples
  - So, we need index
- We want to avoid sorting – read <right> tuples in <right> order
  - So, we need special strategy to traverse index
- We want to support tuples visibility
  - So, we should be able to resume index traverse
- We want to support many data types
  - So, we need to modify GiST
• Synthetic data – randomly distributed points

```sql
create table qq ( id serial, p point, s int4);
insert into qq (p,s) select point( p.lat, p.long), (random()*1000)::int
from ( select (0.5-random())*180 as lat, random()*360 as long
    from ( select generate_series(1,1000000) ) as t
) as p;
create index qq_p_s_idx on qq using gist(p);
analyze qq;
```

• Query – find k-closest points to (0,0)

```sql
set enable_indexscan=on|off;
explain (analyze on, buffers on)
    select * from qq order by (p <-> '(0,0)') asc limit 10;
```
Knn-search: Examples

• **postgresql.conf:**

  
  ```
  shared_buffers = 512MB #32MB  
  work_mem = 32MB #1MB  
  maintenance_work_mem = 256MB #16MB  
  checkpoint_segments = 16  
  effective_cache_size = 1GB #128MB  
  ```

• **Index statistics (n=1000,000)**

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of levels</td>
<td>3</td>
</tr>
<tr>
<td>Number of pages</td>
<td>8787</td>
</tr>
<tr>
<td>Number of leaf pages</td>
<td>8704</td>
</tr>
<tr>
<td>Number of tuples</td>
<td>1008786</td>
</tr>
<tr>
<td>Number of invalid tuples</td>
<td>0</td>
</tr>
<tr>
<td>Number of leaf tuples</td>
<td>1000000</td>
</tr>
<tr>
<td>Total size of tuples</td>
<td>44492028 bytes</td>
</tr>
<tr>
<td>Total size of leaf tuples</td>
<td>44104448 bytes</td>
</tr>
<tr>
<td>Total size of index</td>
<td>71983104 bytes</td>
</tr>
</tbody>
</table>
k=1, n=1,000,000

Limit (cost=24853.00..24853.00 rows=1 width=24) (actual time=469.129..469.130 rows=1 loops=1)
  Buffers: shared hit=7353
  ->  Sort (cost=24853.00..27353.00 rows=1000000 width=24) (actual time=469.128..469.128 rows=1 loops=1)
      Sort Key: ((p <-> '(0,0)'::point))
      Sort Method: top-N heap sort Memory: 25kB
      Buffers: shared hit=7353
      ->  Seq Scan on qq (cost=0.00..19853.00 rows=1000000 width=24)
          (actual time=0.007..241.539 rows=1000000 loops=1)
              Buffers: shared hit=7353
  Total runtime: 469.150 ms

Limit (cost=0.00..0.08 rows=1 width=24) (actual time=0.104..0.104 rows=1 loops=1)
  Buffers: shared hit=4
  ->  Index Scan using qq_p_idx on qq (cost=0.00..82060.60 rows=1000000 width=24)
      (actual time=0.104..0.104 rows=1 loops=1)
      Sort Cond: (p <-> '(0,0)'::point)
      Buffers: shared hit=4
  Total runtime: 0.117 ms 4000 times faster!
### knn-search: Examples

\( n = 1000,000 \)

<table>
<thead>
<tr>
<th>k</th>
<th>hit</th>
<th>knn</th>
<th>seq</th>
<th>sortmem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>0.117</td>
<td>469.150</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>17</td>
<td>0.289</td>
<td>471.735</td>
<td>25</td>
</tr>
<tr>
<td>100</td>
<td>118</td>
<td>0.872</td>
<td>468.244</td>
<td>32</td>
</tr>
<tr>
<td>1000</td>
<td>1099</td>
<td>7.107</td>
<td>473.840</td>
<td>127</td>
</tr>
<tr>
<td>10000</td>
<td>10234</td>
<td>31.629</td>
<td>525.557</td>
<td>1550</td>
</tr>
<tr>
<td>100000</td>
<td>101159</td>
<td>321.182</td>
<td>994.925</td>
<td>13957</td>
</tr>
</tbody>
</table>
## Knn-search: Examples

For $n = 10,000$: 

<table>
<thead>
<tr>
<th>K</th>
<th>hit</th>
<th>:kn n</th>
<th>: seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>:3</td>
<td>0.117</td>
<td>6.072</td>
</tr>
<tr>
<td>10</td>
<td>:13</td>
<td>0.247</td>
<td>5.014</td>
</tr>
<tr>
<td>100</td>
<td>:103</td>
<td>0.295</td>
<td>6.381</td>
</tr>
<tr>
<td>1000</td>
<td>:996</td>
<td>1.605</td>
<td>8.670</td>
</tr>
<tr>
<td>10000</td>
<td>:9916</td>
<td>16.487</td>
<td>14.706</td>
</tr>
</tbody>
</table>

$k$nn lose if $k = n$, $n$ is small
- Real data
- 2 mln points
- US, geonames
• Query: find 10 closest points in US with 'mars' in names to the point (5,5) - create composite index:

```sql
create index pt_fts_idx on geo using gist(point, to_tsvector('english', asciiname));

=# explain (analyze on, buffers on) select asciiname, point, (point <-> '5.0,5.0'::point) as dist from geo where to_tsvector('english', asciiname) @@ to_tsquery('english', 'mars') order by dist asc limit 10;
```

```sql
QUERY PLAN

Limit  (cost=0.00..33.55 rows=10 width=35) (actual time=0.452..0.597 rows=10 loops=1)
  Buffers: shared hit=56
  ->  Index Scan using pt_fts_idx on geo  (cost=0.00..34313.91 rows=10227 width=35) (actual time=0.452..0.592 rows=10 loops=1)
        Index Cond: (to_tsvector('english'::regconfig, (asciiname)::text) @@ ''mar''::tsquery)
        Sort Cond: (point <-> '(5,5)'::point)
        Buffers: shared hit=56
  Total runtime: 0.629 ms
(7 rows)
```
### knn-search: Existing solutions

#### Select id, date, event from events order by date <-> '1957-10-04':'::date asc limit 10;

<table>
<thead>
<tr>
<th>id</th>
<th>date</th>
<th>event</th>
</tr>
</thead>
<tbody>
<tr>
<td>58137</td>
<td>1957-10-04</td>
<td>U.S.S.R. launches Sputnik I, 1st artificial Earth satellite</td>
</tr>
<tr>
<td>58136</td>
<td>1957-10-04</td>
<td>&quot;Leave It to Beaver,&quot; debuts on CBS</td>
</tr>
<tr>
<td>117062</td>
<td>1957-10-04</td>
<td>Gregory T Linteris, Demarest, New Jersey, astronaut, sk: STS 83</td>
</tr>
<tr>
<td>117061</td>
<td>1957-10-04</td>
<td>Christina Smith, born in Miami, Florida, playmate, Mar, 1978</td>
</tr>
<tr>
<td>102670</td>
<td>1957-10-05</td>
<td>Larry Saumell, jockey</td>
</tr>
<tr>
<td>31456</td>
<td>1957-10-03</td>
<td>Willy Brandt elected mayor of West Berlin</td>
</tr>
<tr>
<td>58291</td>
<td>1957-10-05</td>
<td>12th Ryder Cup: Britain-Ireland, 7 -4 at Lindrick GC, England</td>
</tr>
<tr>
<td>58290</td>
<td>1957-10-05</td>
<td>11th NHL All-Star Game: All-Stars beat Montreal 5-3 at Montreal</td>
</tr>
<tr>
<td>58292</td>
<td>1957-10-05</td>
<td>Yugoslav dissident Milovan Djilos sentenced to 7 years</td>
</tr>
<tr>
<td>102669</td>
<td>1957-10-05</td>
<td>Jeanne Evert, tennis player, Chris' sister</td>
</tr>
</tbody>
</table>

(10 rows)

Time: 115.548 ms

#### Very inefficient:
- Full table scan, btree index on date won't help.
- Sort full table
### contrib/btree_gist

```sql
knn=# select id, date, event from events order by date <-> '1957-10-04'::date asc limit 10;
```

<table>
<thead>
<tr>
<th>id</th>
<th>date</th>
<th>event</th>
</tr>
</thead>
<tbody>
<tr>
<td>58137</td>
<td>1957-10-04</td>
<td>U.S.S.R. launches Sputnik I, 1st artificial Earth satellite</td>
</tr>
<tr>
<td>58136</td>
<td>1957-10-04</td>
<td>&quot;Leave It to Beaver,&quot; debuts on CBS</td>
</tr>
<tr>
<td>117062</td>
<td>1957-10-04</td>
<td>Gregory T Linteris, Demarest, New Jersey, astronaut, sk: STS 83</td>
</tr>
<tr>
<td>117061</td>
<td>1957-10-04</td>
<td>Christina Smith, born in Miami, Florida, playmate, Mar, 1970</td>
</tr>
<tr>
<td>102670</td>
<td>1957-10-05</td>
<td>Larry Saumell, jockey</td>
</tr>
<tr>
<td>31456</td>
<td>1957-10-03</td>
<td>Willy Brandt elected mayor of West Berlin</td>
</tr>
<tr>
<td>58291</td>
<td>1957-10-05</td>
<td>12th Ryder Cup: Britain-Ireland, 7 -4 at Lindrick GC, England</td>
</tr>
<tr>
<td>58290</td>
<td>1957-10-05</td>
<td>11th NHL All-Star Game: All-Stars beat Montreal 5-3 at Montreal</td>
</tr>
<tr>
<td>58292</td>
<td>1957-10-05</td>
<td>Yugoslav dissident Milovan Djilos sentenced to 7 years</td>
</tr>
<tr>
<td>102669</td>
<td>1957-10-05</td>
<td>Jeanne Evert, tennis player, Chris' sister</td>
</tr>
</tbody>
</table>

(10 rows)

Time: 0.590 ms

- **Very inefficient:**
  - 8 index pages read + 10 tuples read
  - NO sorting
  - About 200 times faster !
pg_trgm support – distance = 1 – Similarity

knn=# select date, event, ('jeorge ewashington' <-> event) as dist from events order by dist asc limit 10;

<table>
<thead>
<tr>
<th>date</th>
<th>event</th>
<th>dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>1732-02-11</td>
<td>George Washington</td>
<td>0.458333</td>
</tr>
<tr>
<td>1792-12-05</td>
<td>George Washington re-elected U.S. pres</td>
<td>0.674419</td>
</tr>
<tr>
<td>1811-02-23</td>
<td>George Washington Hewitt, composer</td>
<td>0.675</td>
</tr>
<tr>
<td>1753-08-04</td>
<td>George Washington becomes a master mason</td>
<td>0.697674</td>
</tr>
<tr>
<td>1941-07-19</td>
<td>Jennifer Dunn, Rep-R-Washington</td>
<td>0.710526</td>
</tr>
<tr>
<td>1945-05-12</td>
<td>Jayotis Washington, rocker</td>
<td>0.714286</td>
</tr>
<tr>
<td>1817-05-05</td>
<td>George Washington Julian, MC, Union, died in 1899</td>
<td>0.72549</td>
</tr>
<tr>
<td>1789-08-25</td>
<td>Mary Ball Washington, mother of George, dies</td>
<td>0.729167</td>
</tr>
<tr>
<td>1844-01-12</td>
<td>George Washington Cable, American Novelist</td>
<td>0.729167</td>
</tr>
<tr>
<td>1925-01-31</td>
<td>George Washington Cable, American Novelist</td>
<td>0.729167</td>
</tr>
</tbody>
</table>

(10 rows)

Time: 187.604 ms
Corner case for knn-search - all data are on the same distance from point Q!
• Corner case for Best First Strategy - all data are on the same distance from point Q!

cREATE TABLE circle (id serial, p point, s int4);
INSERT INTO circle (p, s)
    SELECT point(p.x, p.y), (random()*1000)::int
    FROM (SELECT t.x, sqrt(1 - t.x*t.x) AS y
          FROM (SELECT random() AS x, generate_series(1,1000000) AS t)
          AS p)
    AS p;
CREATE INDEX circle_p_idx ON circle USING gist(p);
analyze circle;

Number of levels: 3
Number of pages: 8266
Number of leaf pages: 8201
Knn-search: Examples

- Corner case for knn-search - all data are on the same distance from point Q!

```sql
=# explain (analyze on, buffers on) select * from circle
   order by (p <-> '(0,0)') asc limit 10;
```

```
Limit  (cost=0.00..0.80 rows=10 width=24) (actual time=226.907..226.924 rows=10 loops=1)
   Buffers: shared hit=8276
   ->  Index Scan using circle_p_idx on circle  (cost=0.00..79976.58 rows=1000000 width=24) (actual time=226.905..226.921 rows=10 loops=1)
       Sort Cond: (p <-> '(0,0)::point')
       Buffers: shared hit=8276 - read all index
Total runtime: 230.885 ms
```

- Still 2 times faster than SEQ (454.331 ms) because of sorting
Committed to 9.1
• Начало проекта - Sep 8, 2007 at 7:54 PM

date Sat, Sep 8, 2007 at 7:54 PM
subject Chat with Sergey V. Karpov

7:36 PM me: я тут knn-search занимаюсь, масса интересного. Все думаю, как в постгресе это поиметь
Sergey: а что это такое?
7:37 PM me: k-nearest соседей - супер важная задача
    найти 5 ближайших точек
7:38 PM Sergey: ближайших к чему?
    me: к заданной точке
7:39 PM Sergey: в какой системе координат?
    me: в любой, в n-мерном пространстве. В простом варианте - хотя бы на земле/небе
7:40 PM это нужно для поиска похожих картинок, например.
    навиный вариант повторять запросы - не катит
Knn: History of development

- TODO (http://www.sai.msu.su/~megera/wiki/TODO) начало 2008 года, уже есть понимание что делать
- 25 июля 2009 года – письмо Paul Ramsey (POSTGIS)
- 10 июля 2009 года – контракт Open Planning Project Inc.
- 20 ноября 2009 года – патч KNNGiST v.0.1 (модуль расш)
- Commitfest nightmare
  - 22 июля 2010 – KNNGiST (v.0.8), commitfest
  - 13 сентября 2010 – KNNGiST (v.0.9)
  - 03 декабря 2010 – Tom Lane committed for 9.1!
  - 21 января 2011 – contrib/btree_gist committed!
• Итого: На проект ушло больше 3 лет!
• Реальное программирование заняло несколько месяцев
• Основные причины:
  • Отсутствие поддержки
    – Слабая информационная связь разработчиков и заказчиков
  • Занятость разработчиков
  • Усложнение процедуры рассмотрения проектов в сообществе