Interval binning Documentation

Release 1.0.0

Martijn Vermaat

December 22, 2015

Contents

These are some utility functions for working with the interval binning scheme as used in the UCSC Genome Browser. This scheme can be used to implement fast overlap-based querying of intervals, essentially mimicking an R-tree index.

Note: Some database systems natively support spatial index methods such as R-trees. See for example the PostGIS extension for PostgreSQL.

Although in principle the method can be used for binning any kind of intervals, be aware that the largest position supported by this implementation is 2^{29} (which covers the longest human chromosome).

CHAPTER 1

Usage

Let's say you have a set of intervals I in a database system without support for spatial indexing. Querying I on overlap with an interval q can be done as:

$$\{i \in I \mid \text{overlapping}(i, q)\}$$

where

$$\text{overlapping}(i, q) = i.start < q.stop \wedge i.stop > q.start$$

But this will be slow, even with normal B-tree indexes on *start* and *stop*.

If for each interval *i*, we also store its bin as given by `assign_bin()` (and we index it), we can get the same result much faster by pre-filtering on `overlapping_bins()`:

$$\{i \in I \mid i.bin \in \text{overlapping\_bins}(q) \wedge \text{overlapping}(i, q)\}$$

Similarly, if i must completely contain q (or vice versa), you can use `containing_bins()` (or `contained_bins()`).

1.1 SQLAlchemy example

As a more concrete example, let's consider the following SQLAlchemy model definition for storing gene locations:

```
class Gene(Base):
    name = Column(String, primary_key=True)
    start = Column(Integer)
    stop = Column(Integer)
    bin = column(Integer, index=True)

    def __init__(self, name, start, stop):
        self.name = name
        self.start = start
        self.stop = stop
        self.bin = binning.assign_bin(start, stop)
```

The *bin* column is populated using `assign_bin()` and has an index. We can ask for all genes spanning the 50,000-50,500 interval:

```
>>> session.query(Gene).filter(Gene.start < 50000,
...                            Gene.stop > 50500).count()
78
```

But that query will be slow, yielding a sequential table scan. Adding indexes on *Gene.start* or *Gene.stop* will not help much.

The *containing_bins()* function gives us all the bins potentially containing genes spanning some interval. We can use that to filter on *Gene.bin* first:

```
>>> bins = binning.containing_bins(50000, 50500)
>>> session.query(Gene).filter(Gene.bin.in_(bins),
...                            Gene.start < 50000,
...                            Gene.stop > 50500).count()
78
```

This query will be much faster because it can use the index on *Gene.bin*. The filter on bin only gives us a crude pre-selection though, so we still have to apply the *Gene.start* and *Gene.stop* filters on the (relatively small) resulting set of genes to get the exact answer.

CHAPTER 2

Installation

To install the latest release via PyPI using pip:

```
pip install interval-binning
```

The latest development version can be found on GitHub.

CHAPTER 3

API documentation

All positions and ranges in this module are zero-based and open-ended, following standard Python indexing and slicing.

binning.**assign_bin**(*start*, *stop*)
Given an interval *start:stop*, return the smallest bin in which it fits.

Parameters **start, stop** (*int*) – Interval positions (zero-based, open-ended).

Returns Smallest bin containing *start:stop*.

Return type int

Raises **OutOfRangeError** If *start:stop* exceeds the range of the binning scheme.

binning.**overlapping_bins**(*start*, *stop=None*)
Given an interval *start:stop*, return bins for intervals *overlapping start:stop* by at least one position. The order is according to the bin level (starting with the smallest bins), and within a level according to the bin number (ascending).

Parameters **start, stop** (*int*) – Interval positions (zero-based, open-ended). If *stop* is not provided, the interval is assumed to be of length 1 (equivalent to *stop = start + 1*).

Returns All bins for intervals overlapping *start:stop*, ordered first according to bin level (ascending) and then according to bin number (ascending).

Return type list(int)

Raises **OutOfRangeError** If *start:stop* exceeds the range of the binning scheme.

binning.**containing_bins**(*start*, *stop=None*)
Given an interval *start:stop*, return bins for intervals completely *containing start:stop*. The order is according to the bin level (starting with the smallest bins), and within a level according to the bin number (ascending).

Parameters **start, stop** (*int*) – Interval positions (zero-based, open-ended). If *stop* is not provided, the interval is assumed to be of length 1 (equivalent to *stop = start + 1*).

Returns All bins for intervals containing *start:stop*, ordered first according to bin level (ascending) and then according to bin number (ascending).

Return type list(int)

Raises **OutOfRangeError** If *start:stop* exceeds the range of the binning scheme.

binning.**contained_bins**(*start*, *stop=None*)
Given an interval *start:stop*, return bins for intervals completely *contained by start:stop*. The order is according to the bin level (starting with the smallest bins), and within a level according to the bin number (ascending).

Parameters **start, stop** (*int*) – Interval positions (zero-based, open-ended). If *stop* is not provided, the interval is assumed to be of length 1 (equivalent to *stop = start + 1*).

Returns All bins for intervals contained by *start:stop*, ordered first according to bin level (ascending) and then according to bin number (ascending).

Return type list(int)

Raises OutOfRangeError If *start:stop* exceeds the range of the binning scheme.

binning.**covered_interval**(*bin*)

Given a bin number *bin*, return the interval covered by this bin.

Parameters **bin** (*int*) – Bin number.

Returns Tuple of *start, stop* being the zero-based, open-ended interval covered by *bin*.

Return type tuple(int)

Raises OutOfRangeError If bin number *bin* exceeds the maximum bin number.

CHAPTER 4

Copyright

This library is licensed under the MIT License, meaning you can do whatever you want with it as long as all copies include these license terms. The full license text can be found in the LICENSE.rst file.

See the AUTHORS.txt for for a complete list of copyright holders.

b

A

B

C

O