Astronomical Data Analysis Software and Systems XII
ASP Conference Series, Vol. 295, 2003
H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds.

A New Way of Joining Source Catalogs using a Relational Database Management System

Clive G. Page

AstroGrid Project, Department of Physics & Astronomy, University of Leicester, U.K.

Abstract. New science often arises from joining source catalogs obtained from different wavebands, but efficient methods for performing such joins depend on multi-dimensional indexing and are poorly handled by most DBMS. A new algorithm, the PCODE, is proposed which allows any relational DBMS to carry out this operation efficiently.

1. The Fuzzy Join Problem

New science often arises from investigations based on the cross-identification of sources in different wavebands. Since source catalogs are often stored as tables in a relational database management system (DBMS), in theory this just requires a suitable `JOIN` between tables, based on source position. Even if positions alone do not produce unique associations, so that additional criteria are needed, positional coincidence is still the primary key for such joins. In many cases the absence of a counterpart in another band will be scientifically interesting: any unmatched sources can be found using an `OUTER JOIN`. So much for theory: the practice is more complicated.

- A two-dimensional index is needed to handle the (RA, Dec) coordinate pair, but only a few DBMS support multi-dimensional indexing, and for some of these it is an expensive add-on.
- The join also needs a range search rather than an exact match, since positions always have errors. Typically the errors in RA and Dec are similar, so that the error region is a small circle on the sky.
- This type of positional match has come to be called a *fuzzy join*, since the matching criterion is the overlap of the two error-circles, and the distances need to use a great-circle distance function.
- Indices developed for a Cartesian grid need to be modified when applied to a spherical-polar coordinate frame as the scales are distorted, with singularities at the poles (and wrap-around on the RA axis).

Because of these difficulties, many existing data archives index their catalogs on only one spatial coordinate (usually declination); this is adequate for single object searches, but is neither efficient nor scalable for large-scale joins, because the one-dimensional index is selecting a slice all around the celestial sphere, much larger than the region actually required. And every year catalogs get bigger: the recently announced USNO-B1.0 (Monet & Levine, this conference) has over 10^9 rows and several others are nearly as large.

2. Multi-dimension Indexing

Multi-dimensional indexing has been a hot topic of computer science research for many years, and many different algorithms have been proposed. The names include: BANG file, BV-tree, Buddy tree, Cell tree, G-tree, GBD-tree, Gridfile, hB-tree, kd-tree, LSD-tree, P-tree, PK-tree, PLOP hashing, Pyramid tree, Q-tree, Quadtree, R-tree, SKD-tree, SR-tree, SS-tree, TV-tree, UB-tree, and Z-order index.

Whereas the B-tree has become the standard one-dimensional indexing method used in every DBMS, none of the multi-dimensional indices has comparable properties such as fast insertion and retrieval, and a worst-case fill-factor of 50%. The problem is, of course, fundamentally intractable, since one cannot serialise the points on a two-dimensional surface. One of the earliest algorithms, the R-tree (Guttman 1984), has been implemented in a few DBMS, including one of the Open Source products: PostgreSQL. But the R-tree has some limitations for source catalogs:

- R-trees index rectangular areas: since some sources within the rectangle will be outside the enclosed error-circle, these have to be removed in a subsequent filtering stage.
- The width of the boxes gets large close to the poles, eventually spanning the whole range of RA.
- The R-tree stores four coordinates per point so it uses much more disc space than a B-tree.
- The wrap-around of RA at zero hours needs special tests in the code.

Kalpakis et al (2001) used Informix, which also supports R-trees, to index large catalogs, but found that the data loading and index creation times were substantial: they estimated it would take 25 days to create an index for the whole USNO-A1.0 catalog.

3. PCODE Method

An alternative algorithm, proposed here, is based on covering the sky with pixels of approximately equal area, and assigning a unique integer to each pixel. A number of suitable pixelisation schemes have been proposed, the best known being: HTM, the Hierarchical Triangular Mesh, (Kunszt et al. 2000), and HEALPix, the Hierarchical Equal Area iso-Latitude Pixelisation, (Górski et al. 1998). Both pixelisations are suitable for generating PCODE values, and both have selectable resolution.

For each table, the pixel code value corresponding to the central position (RA, Dec) of each source is computed and inserted into a new column called PCODE. A B-tree index is then created on the PCODE column. This cannot be the primary index of a table since PCODE values are not necessarily unique: one pixel may contain more than one source. The two tables can then be joined on their PCODE columns: this is simply an equi-join between integers, which all DBMS handle efficiently. Note that HTM, and one of the two numbering schemes for HEALPix, are designed to assign nearby numbers to adjacent pixels as far as possible, but this property is not used here; indeed indexing schemes based on this, such as the Z-order index, have serious failings.

There are a number of obvious problems with the simple method outlined above, but they can be solved:

1. Two objects may be in the same pixel but do not have overlapping error-circles.
   Solution: filter the results with an additional pass using a great-circle distance function (just as when an R-tree is used).
2. Two sources may have overlapping error circles but centers in different pixels.
   Solution: insert an additional row in the table whenever an error-circle overlaps an additional pixel (this extra row has all the attributes of the original, except for the PCODE value).
3. If two sources both extend into the same two pixels then the product of the join will itself contain this match in duplicate.
   Solution: in the post-filtration stage weed out these duplicates using a `DISTINCT` clause in the SQL. In many cases the tables will already contain a unique identifier, otherwise an addition column with a unique integer sequence number needs to be added.

A single SQL statement can be devised to do all these filtering stages, but the margin is too small to contain it.

4. R-tree versus PCODE

In theory a B-tree index built on PCODEs should be more efficient than an R-tree, and this was confirmed in practice. We took subsets of the HST Guide Star Catalog and the USNO-A1 Catalog corresponding to an area of around 400 square degrees. This produced tables of 275,154 rows and 3,476,948 rows respectively. The HEALPix pixelisation scheme was used, with $N_{side} = 8192$ giving 805,306,368 pixels, each $\sim$26 arc-seconds across, and required only 32-bit PCODE values. The tests used PostgreSQL v7.2.1, since it supports R-trees, and were run on a 450 MHz PC running Linux.

Algorithm	Table size (MB)	Index size (MB)	Index Creation (secs)	Fuzzy Join (secs)
Native R-tree	496	248	6408	347
PCODE and B-tree	293	64	233	143

Not only is the PCODE method faster and smaller, it can also be used with *any* relational database management system, not just the few which support a spatial index.

5. Scalability of PCODE

In order to ensure that only a small proportion of additional rows are needed, one must choose a pixel size larger than the typical error-circle radius. Actually it is a rule of thumb in sky surveys that detections should be limited to the level at which there is no more than one source per 40 beam-areas, otherwise confusion effects dominate. It should, therefore, always be possible to find a pixel resolution which is fine enough to leave fewer than one object per pixel on average, while coarse enough to have relatively few error-circles crossing a pixel boundary. Of course there will always be a few source postions which lie close to the point at which 4 HEALPix pixels (or 6 HTM pixels) touch. Efficiency may be lower when dealing with surveys where the density of sources is very uneven, with many pixels empty, but some holding large numbers of sources (e.g., in the Galactic Plane). The PCODE method is also less suitable if a wide-area of the sky is of interest, e.g., when making a finding-chart.

A difficulty may still arise when joining two tables with very different resolutions, such as an X-ray catalog from the era preceding XMM-Newton and Chandra, and a modern optical catalog. In practice the problem is less serious, as a catalog with large error regions is, intrinsically, one with fewer sources in it, so the absolute number of extra rows needed is tolerable. In fact the 26 arc-second pixels we used in our tests turn out to be a good match to most current catalogs.

There is also a potential scalability limitation from the need to use `SELECT DISTINCT` on the output of the join, since this implies that the results of the join are sorted. This takes a time proportional to $n.log(n)$, where n is the number of rows in the *output* table.

The HTM pixelisation has a useful property which promises additional scalability: the algorithm is recursive so that four pixels at each resolution stage fit exactly in one pixel from the next coarser resolution, and the pixel codes just add extra bits. Hence if a fine mesh is established using a PCODE-equipped table based on the HTM pixelisaton, it can be converted to any required coarser resolution, e.g., for joining with a low-resolution source catalog, just by ignoring pairs of bits at the least-significant end of the integer.

This suggests that all source catalogs could have PCODE values attached, based on HTM at some a suitably high resolution, in order to make a wide range of joins (and cone-searches) efficient. At lower resolution many of the added rows become superfluous but a `SELECT DISTINCT` will remove them too.

References

Górski, K. M., Hivon, E., & Wandelt, B. D. 1998, in Proc MPA/ESO Cosmology Conference on Evolution of Large-scale Structure

Guttman, A. 1984, in Proc. ACM SIGMOD Int. Conf. on Management of Data, 47

Kalpakis, K., Riggs, M., Pasad, M., Puttagunta, V., & Behnke, J. 2001, in ASP Conf. Ser., Vol. 238, Astronomical Data Analysis Software and Systems X, ed. F. R. Harnden, Jr., F. A. Primini, & H. E. Payne (San Francisco: ASP), 133

Kunszt, P. Z., Szalay, A. S., Csabai, I., & Thakar, A. R. 2000, in ASP Conf. Ser., Vol. 216, Astronomical Data Analysis Software and Systems IX, ed. N. Manset, C. Veillet, & D. Crabtree (San Francisco: ASP), 141