Astronomical Data Analysis Software and Systems XX
ASP Conference Series, Vol. 442
Ian N. Evans, Alberto Accomazzi, Douglas J. Mink, and Arnold H. Rots, eds.
© 2011 Astronomical Society of the Pacific

Efficient and Scalable Cross-Matching of (Very) Large Catalogs

François-Xavier Pineau, Thomas Boch, and Sébastien Derriere

CDS, Observatoire Astronomique de Strasbourg, Université de Strasbourg, CNRS, 11 rue de l'Université, 67000 Strasbourg, France

Abstract. Whether it be for building multi-wavelength datasets from independent surveys, studying changes in objects luminosities, or detecting moving objects (stellar proper motions, asteroids), cross-catalog matching is a technique widely used in astronomy. The need for efficient, reliable and scalable cross-catalog matching is becoming even more pressing with forthcoming projects which will produce huge catalogs in which astronomers will dig for rare objects, perform statistical analysis and classification, or real-time transients detection. We have developed a formalism and the corresponding technical framework to address the challenge of fast cross-catalog matching. Our formalism supports more than simple nearest-neighbor search, and handles elliptical positional errors. Scalability is improved by partitioning the sky using the HEALPix scheme, and processing independently each sky cell. The use of multi-threaded two-dimensional *k*d-trees adapted to managing equatorial coordinates enables efficient neighbor search. The whole process can run on a single computer, but could also use clusters of machines to cross-match future very large surveys such as GAIA or LSST in reasonable times. We already achieve performances where the 2MASS (~ 470M sources) and SDSS DR7 (~ 350M sources) can be matched on a single machine in less than 10 minutes. We aim at providing astronomers with a catalog cross-matching service, available on-line and leveraging on the catalogs present in the VizieR database. This service will allow users both to access pre-computed cross-matches across some very large catalogs, and to run customized cross-matching operations. It will also support VO protocols for synchronous or asynchronous queries.

Introduction

The largest catalogs of astronomical sources built so far, e.g., the USNOB1, contain about one billion sources. Projections for the LSST anticipate a number of unique sources about three times greater after 5 years of exploitation. With a minimum of 6 parameters — identifier (integer), positions (doubles) and associated errors (floats) — it will represent about 100 GB of data.

The method used to cross-correlate such catalogs has to take into account the current trend in computer hardware improvements: increasing and faster memory, more cores but stable clock frequency, cheaper machines grouped in clusters. Therefore it has to be scalable with its performance depending on both available machines and individual process efficiency. A catalog cross-match task must thus be split into pieces of various sizes, which can be processed independently on different threads (multi-threading) distributed on different machines (parallel processing).

This article is organized as following: § 1 deals with the partitioning and multi-threading of a cross-match task, § 2 with modified *k*d-tree for counterparts searches, § 3 with data loading, and the § 4 presents some results.

1. HEALPix Partitioning and Multi-Threaded Pixel Processing

HEALPix (Górski et al. 2005) is a hierarchical sky partitioning developed at NASA. At level 0, the sky is divided into 12 pixels. Then, at each successive level, the pixels are divided into four new pixels so that for a given level l, the number of pixels is $N_l = 12 \times 4^l$.

We use HEALPix to divide the sky into pixels that can be processed independently, one by one on a single machine and simultaneously on a cluster of machines. The chosen HEALPix-pixels level depends on several parameters, such as the density of sources in both catalogs and the available memory. To cross-match the sources contained in a pixel of a catalog A with a catalog B, we first load the sources of the catalog B which are in the pixel. In order not to miss some correlations, we also have to load catalog B sources which are in an extra border around the pixel (see Fig. 1). We then build a modified 2d-tree (see § 2) containing the sources

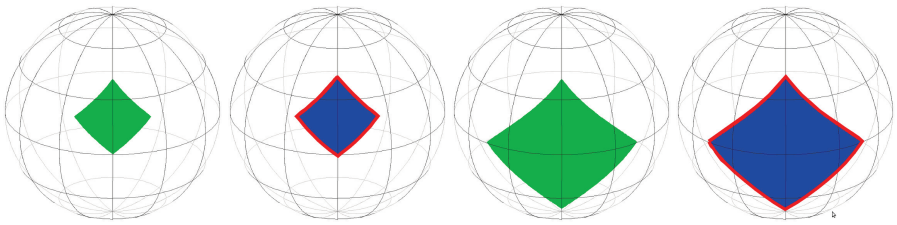

Figure 1. HEALPix pixels of level 1 and 0 for a two catalogs A and B. No to miss some correlations, an extra border is required around the catalog B pixels.

retrieved from the catalog B. Then, for each catalog A source in the pixel, we perform a cone-search query in the 2d-tree. The process is multi-threaded: we create a pool of threads and, until all sources have been processed, we 1) take a worker from the pool, 2) fill the worker queue with catalog A sources, 3) wake-up the worker thread for correlation and result writing and 4) put back the worker in the pool, waiting for new sources to correlate.

2. Multi-Threaded Modified *k*d-tree

A *k*d-tree is an Euclidean space-partitioning data structure especially adapted for fast *k*-nearest-neighbor (*k*NN) queries. We want the lightest possible data structure storing spherical coordinates and allowing us to perform fast cone-search or *k*NN queries relying on angular distances. The solution we have developed and adopted is a 2 dimensional *k*d-tree (2d-tree) stored in an array for which we have modified the query algorithm. The two dimensions are the spherical coordinates, α and δ. The creation of the tree is standard (see Fig. 2): the algorithm is a simple *quicksort* with alternating sorted coordinate.

In a standard fixed radius 2d-tree query, the algorithm first goes down the tree to the leaf node containing the target. It then backs up, and at each parent node it decides to go down the other sub-tree if the disk — defined by the target and the radius of the query — overlaps the rectangular area covered by the sub-tree (see Moore (1991) for more details). With spherical coordinates, the area covered by a sub-tree is no longer rectangular, and the distance between the target and a node is not Euclidean. We thus had to change the standard algorithm by computing target–node angular distances — resorting to the Haversine formula — and implementing a boolean function testing if a cone overlaps a range in α and δ.

The generation of a *k*d-tree is quite straightforward to multi-thread since each sub-tree is built independently (see Fig. 2). Nevertheless, resorting to a multi-threaded sort algorithm for the first nodes would even accelerate the process.

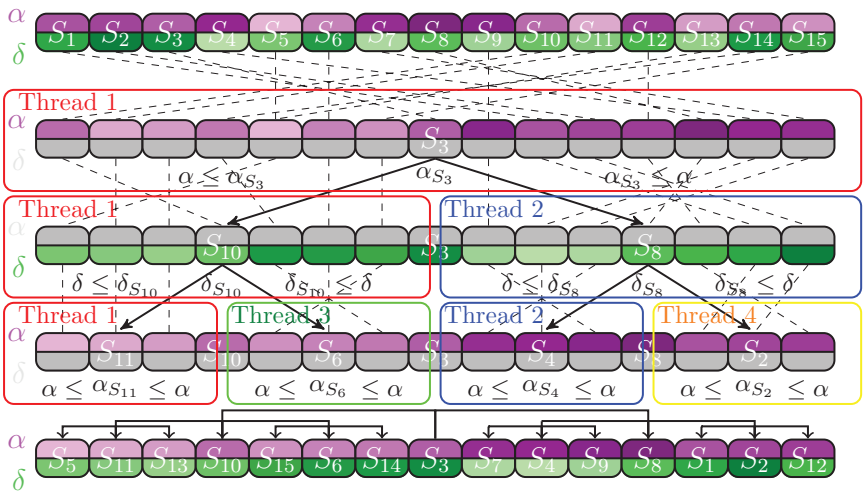

Figure 2. Multi-threaded creation of a 15 sources array 2d-tree.

3. HEALPix Indexed Binary File

An efficient cross-match requires an efficient way to retrieve the needed data: positions, positional errors if necessary, and possibly identifiers. Disk accesses are expensive operations and to avoid too much head movement overhead it is better to access unfragmented data. The basic ideas to optimize data loading are to read only the necessary data, to read them from contiguous blocks, and in a binary format to avoid conversions. For fast access to the different sky cells' contents, data have to be grouped by HEALPix pixels and the file must be indexed.

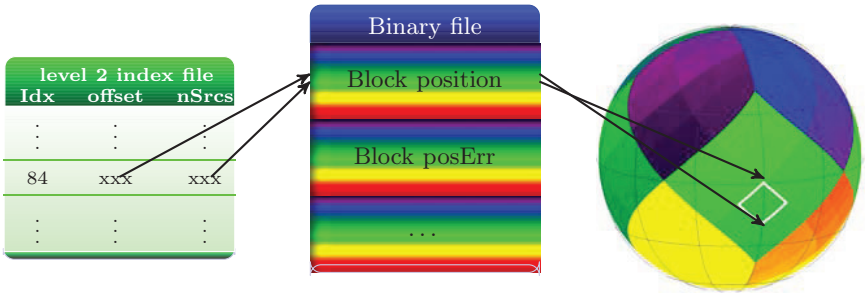

Figure 3. Indexed binary catalog file.

We have implemented an indexed binary file format which stores a catalog by blocks (Fig. 3). Each catalog file contains one block for identifiers, one block for positions, one block for positional errors, and some other blocks. In each block, rows are sorted by the HEALPix pixel indices of the sources they belong to, so that the pixels of different HEALPix levels are stored contiguously in each block.

For each HEALPix level, for all pixels, an index file stores the index of the first row and the number of rows the pixel contains. So for each block, a pixel on the sky maps to a contiguous portion of the file, which is known perfectly thanks to the index files.

4. Performances Tests

We have performed some tests on several large catalogs: SDSS DR7 (~350M sources), 2MASS (~470M sources), and USNOB1 (~1G sources).

Our code is full Java and tests have been performed on a unique machine running Ubuntu 10.04 with Java 1.6.0_20 and a sun 64-Bit JVM (Java Virtual Machine). The machine is a Dell server with 24 GB of 1333 MHz RAM, two hyper-threaded Intel Xeon quad-cores at 2.27 GHz (16 threads available) and a 10 000 rpm HDD. Results are presented Table 1.

Table 1. Results of large catalogs cross-correlation tests. d is the cone-search aperture and d_σ the distance in sigma when taking into account individual elliptical errors on positions.

Catalogs	d	d_σ	nMatch	exec. time
SDSS7 vs 2MASS	5″		49.2 M	~9 min
SDSS7 vs 2MASS	5″	3.44	37.5 M	~10 min
2MASS vs USNOB1	5″		583.3 M	~30 min

All tests have been performed for a HEALPix level 3 (see Górski et al. 2005, Table 1) with border pixels of level 9. The execution time includes: data loading, tree creation, cross-correlation with (d_σ not empty) or without individual elliptical errors on positions, writing of a join file containing for each association the two sources identifiers and the cross-match distance. The candidate selection criteria when using positional errors is described in Pineau et al. (2010). On a similar machine with 2 hyper-threaded six-cores (24 threads available) the SDSS7 versus 2MASS cross-correlation execution time is under 7 minutes.

References

Górski, K. M., Hivon, E., Banday, A. J., Wandelt, B. D., Hansen, F. K., Reinecke, M., & Bartelmann, M. 2005, ApJ, 622, 759

Moore, A. W. 1991, An introductory tutorial on kd-trees, Tech. Rep. Technical Report No. 209, Computer Laboratory, University of Cambridge, Carnegie Mellon University, Pittsburgh, PA

Pineau, F.-X., Motch, C., Carrera, F., Della Ceca, R., Derriere, S., Michel, L., Schwope, A., & Watson, M. G. 2010, ArXiv e-prints. 1012.1727