Serge Monkewitz, IPAC/Caltech Sherry Wheelock, IPAC/Caltech

WAX

A High Performance Spatial Auto-Correlation Application

Description

Associates multi-epoch source observations in massive astronomical databases
Used for 2MASS Merged Source databases, but generally applicable (e.g. to WISE)

The publicly released 2MASS catalogs were generated by running source extraction software on raw image data from many roughly rectangular regions on the sky. These regions often overlap, producing multiple extractions (apparitions) of a single astronomical source. In such situations, 2MASS chose a "best" apparition to include in the source catalogs. Consequently, there are many extractions for which data is not publicly available and astronomers are unable to identify "best" apparitions with algorithms of their choice.

The WAX software addresses these issues by generating groups of apparitions likely to correspond to single astronomical sources. These groups are made available as a catalog along with the complete database of extracted apparitions. The grouping algorithm employed is conservative in the sense that if an apparition is assigned to more than one group, then that apparition and the groups containing it are flagged as confused, but no attempt to resolve confusion is made (information loss is avoided).

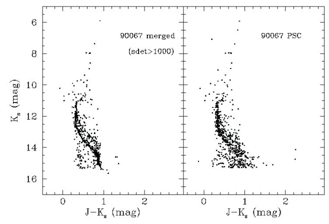
WAX is written in portable C/C++, but has only been run on the Solaris SPARC platform. All I/O is performed directly from/to an RDBMS (currently limited to Informix), thereby saving disk space and database loading time, and allowing output tables to be queried and served immediately.

The grouping (swiss cheese) algorithm is hardwired, and places minimal constraints on the input DBMS table: a per-apparition unique identifier and position suffice to run WAX. The output group catalog will always contain a unique identifier, apparition count, and confusion flag for each group. Also, group/apparition identifier pairs are stored in a link catalog, allowing for the retrieval of all apparitions belonging to a group, or of all groups containing an apparition. Finally, it is important to note that WAX can process subsections of the entire sky, either serially or in parallel.

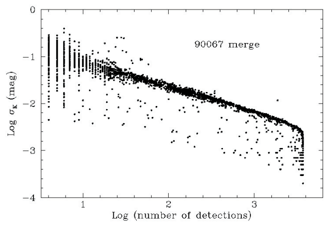
WAX plug-ins can be written to compute arbitrary group attributes (e.g. average position/magnitude) and to choose a "best" group for apparitions belonging to more than one. At compile-time, a plug-in provides a retrieval and output column specification from which C structures corresponding to the columns are generated. This compile-time knowledge of columns is also used to generate high performance database IO code (ESQL/C). At run-time, each generated group and each grouped apparition is passed to the plug-in implementation via the generated data structures. This hides complexity from the plug-in and results in a simple interface (described in a single C header file). As well, libraries are provided for common plug-in tasks such as computing the observational coverage and spatial index of a position.

Taken together, these features allow WAX to be tailored to an apparition database, and extend its usefulness to missions beyond 2MASS.

Below are figures from a 2MASS analysis of WAX output for both point and calibration sources. A pre-filtered copy of the 2MASS Working Point Source Database (~800 million apparitions) was processed by 4 instances of WAX running in parallel for roughly 4.5 days (~1.1 million seconds total CPU time).

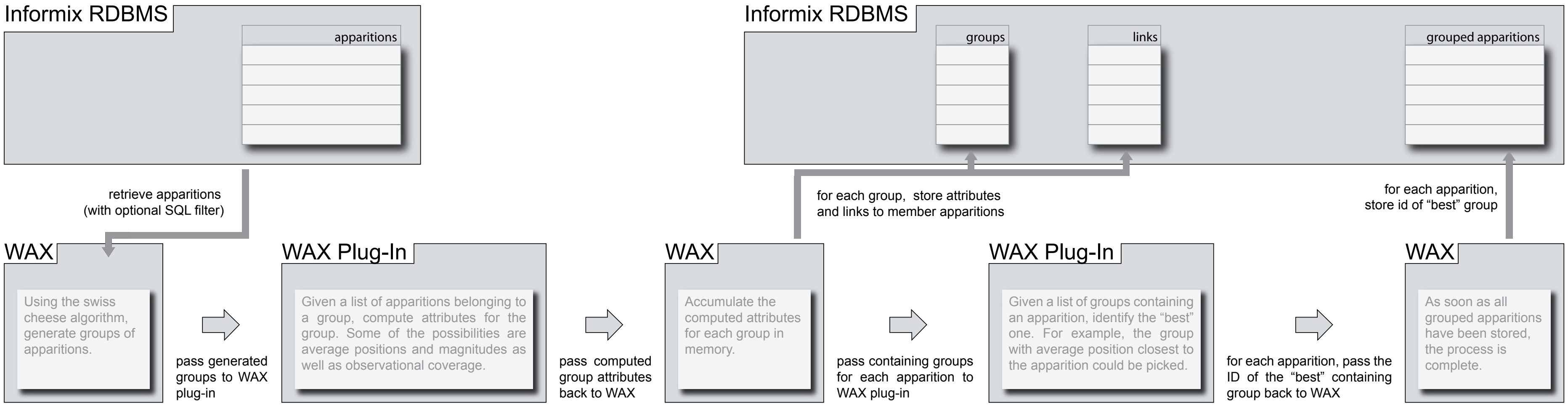


A side-by-side comparison of the color-magnitude diagrams (CMDs) for stars in the 90067 calibration field from the merged WAX output and the single-scan depth 2MASS Point Source Catalog. The left panel shows the CMD from the merged source information table for all sources containing apparitions from 1000 or more scans. The right panel shows the CMD formed from the Point Source Catalog in the same region.



The standard deviation of the mean 2MASS K_s magnitude plotted as a function of the number of K_s detections for merged sources in the 90067 calibration field. The slope of the main locus of points verifies that the standard deviation decreases as the square root of the number of detections, as expected for Gaussian statistics. The vertical spread at any given number of detections is caused by the brightness distribution of the sources - brighter sources have smaller intrinsic uncertainties. This illustrates that the merge has successfully combined measurements of the same source, and not constructed false matches.

Architecture



Informix RDBMS

apparitions

retrieve apparitions
(with optional SQL filter)

WAX

Using the swiss cheese algorithm, generate groups of apparitions.

pass generated groups to WAX plug-in

WAX Plug-In

Given a list of apparitions belonging to a group, compute attributes for the group. Some of the possibilities are average positions and magnitudes as well as observational coverage.

pass computed group attributes back to WAX

WAX

Accumulate the computed attributes for each group in memory.

pass containing groups for each apparition to WAX plug-in

WAX Plug-In

Given a list of groups containing an apparition, identify the "best" one. For example, the group with average position closest to the apparition could be picked.

for each apparition, pass the ID of the "best" containing group back to WAX

WAX

As soon as all grouped apparitions have been stored, the process is complete.

for each group, store attributes and links to member apparitions

for each apparition, store id of "best" group

Informix RDBMS

groups

links

grouped apparitions

Algorithms

Swiss Cheese Algorithm

The swiss cheese algorithm first computes both a *density* and a *centroid* for each apparition on the sky. The *density* for an apparition *a* is defined as the number of apparitions within a specified angular distance *d* of *a*. The *centroid* is defined as the average position of all the apparitions contributing to the *density* of *a*.

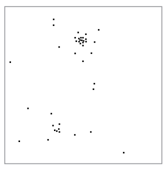
Next, the apparitions are sorted into decreasing *density* order. Each apparition in the resulting queue is considered to be a *seed*; that is, an unprocessed apparition from which a group can potentially be generated. Starting with the apparition at the head of the queue (the *densest seed*), groups are generated :

▪ All apparitions within a specified angular distance *r* of the *seed centroid* are assigned to a new group.

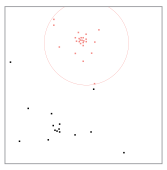
▪ The grouped apparitions are removed from the queue of unprocessed apparitions. Note that the head of the queue will always be removed.

▪ If the queue is non-empty, the process is repeated with the new queue-head.

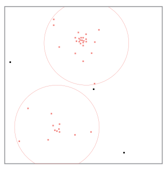
In some circumstances, the algorithm assigns a given apparition to more than one group. Apparitions belonging to two or more groups (and the groups containing them) are said to be *confused*. The diagrams to the right present a step by step illustration of the swiss cheese algorithm and how confusion arises.



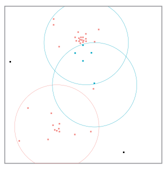
A small region of the sky - unprocessed apparitions are drawn in black.



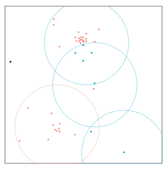
The first group is generated - unconfused apparitions and groups are drawn in pink.



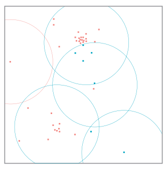
Even after generation of the second group, there is no confusion.



The third group generated introduces confusion, drawn in blue.



The fourth group further confuses the picture.



The final group contains only one apparition - processing for this region is now complete.

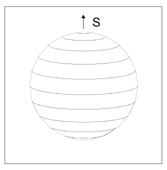
Since the apparition databases being processed are far too large to fit in memory, a straightforward implementation of the algorithm is difficult at best. The WAX software partitions the sky into declination bands to reduce the working set of apparitions, and, instead of processing *seeds* in density order, performs multiple passes over the *seeds* in spatial order. This is done to allow for efficient computation of *densities*, *centroids*, and groups, each of which involve finding apparitions within a small radius of a position. Every pass over the set of *seeds* will only generate groups and discard *seeds* if doing so does not violate the density ordering constraints imposed by the swiss cheese algorithm.

Because WAX cannot consider the sky in its entirety, it is possible for spatial traversals of a single declination band to result in an impasse; that is, a situation in which the generation of any group would produce results inconsistent with those of the swiss cheese algorithm. In such cases, WAX attempts to minimize the deviation incurred.

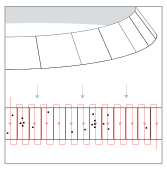
The diagrams to the right depict the spatial subdivision scheme and traversal order used by WAX, and demonstrate how apparitions are processed out of density order:

▪ A group is generated around a *seed centroid* if and only if the *seed* would not be included in any group generated from a *denser seed*.

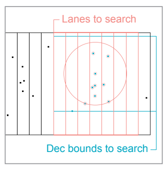
▪ If a group is generated around a *seed*, then the only apparitions in the group that are removed from the queue are those less *dense* than the *seed*.



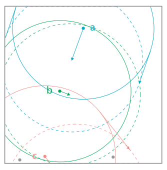
The entire sky, subdivided into declination bands.



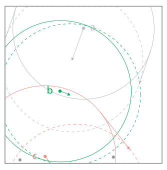
Each band is subdivided into lanes and traversed in spatial order (shown by the pink line).



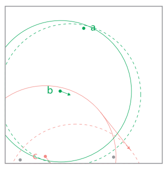
Only a small number of apparitions are considered to find a *density*, *centroid* or group.



Seed a is less dense than *seed* b and contained in the group generated from b. Therefore a is skipped.



The group generated from c does not include b, so a group can be generated from b even though c is denser than b.

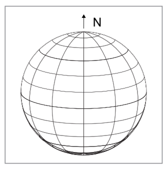


Apparitions a, b, and c are in the group generated from b : c is denser than b and remains in the processing queue.

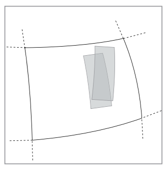
Scan Coverage Algorithm

A *scan* is a quadrilateral on the unit-sphere approximating an area for which raw image data is available and has been used for apparition extraction. The *scan* coverage algorithm determines how many such *scans* cover a particular position on the sky; that is, it determines how many apparitions of an astronomical source to expect at a given position.

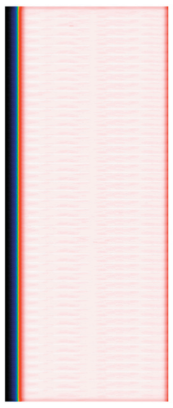
The sky is first subdivided into coarse bins which cover small equal width ranges in right ascension and declination. By testing a position only against the *scans* overlapping the bin covering the position, many coverage tests are avoided. To further improve efficiency, a binary space partitioning tree (BSP tree) is generated offline from the set of *scans* overlapping each bin, and is traversed at runtime for coverage computations.



The entire sky, subdivided into bins.

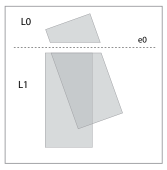


Scans overlapping a single bin.

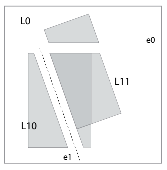


Depth of coverage map for the 2MASS K_s band in calibration field 90067 (frames from north going scans only). The deepest coverage is indicated in white, the shallowest in black.

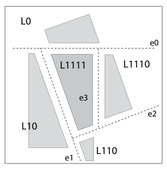
BSP trees are created from a set of *scans* by recursively splitting the set against *scan* edges. The process, illustrated below, produces a binary tree in depth first order, where nodes correspond to splitting planes and leaves are convex polygons inside of which *scan* coverage is constant (no leaf contains a *scan* edge).



The *scans* are split by the first *scan* edge, e0.

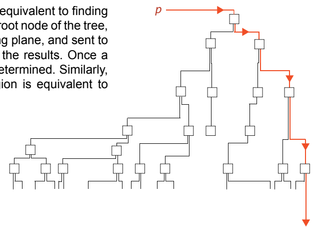


Scan edge e2 is used to split scan set L1.



Two diagram after two more *scan* edge splits.

Computing the *scan* coverage for a position *p* is equivalent to finding the BSP tree leaf containing *p*. Starting with the root node of the tree, *p* is compared against the current node's splitting plane, and sent to the inside or outside child node depending on the results. Once a leaf is reached, *scan* coverage for *p* has been determined. Similarly, computing the *scan* coverage for a search region is equivalent to finding all BSP leaves overlapping the region.



For more information see:
http://irsa.ipac.caltech.edu/applications/2MASS/WAX/docs/html