NACluster: A Non-Supervised Clustering Algorithm for Matching Multi Catalogues

Vinicius P. Freire, José A. F. De Macêdo, Fábio Porto, Reza Akbarinia

NACluster: A Non-Supervised Clustering Algorithm for Matching Multi Catalogues

Vinícius P. Freire and José A. F. de Macêdo
Federal University of Ceará
Fortaleza, Brazil
{vinipires,jose.macedo}@lia.ufc.br

Fábio Porto
National Laboratory of Scientific Computing (LNCC) - DEXL Lab
Petropólis, Brazil
fporto@lncc.br

Reza Akbarinia
INRIA and LIRMM
Montpellier, France
reza.akbarinia@inria.fr

***Abstract*—Astronomy surveys use powerful instruments to browse the sky and identify objects of interest within the surveyed region. Sky objects are individually characterized with spatial coordinates, identifying their position in the sky, in addition to other descriptive attributes. Composing an integrated view of the sky based on catalogues produced by different surveys faces a hard problem of matching objects that have been captured in various catalogues. Due to variations on capturing instruments calibration, the sky position of a single sky object may vary from a catalog to the other. Moreover, in particular dense regions of the sky this problem is exacerbated by a huge number of candidate matches for each given object. Traditional approaches for dealing with this problem use a threshold distance of ϵ to reduce the number of matching candidates. Additionally, they adopt a pairwise approach for matching *n* catalogues inferring transitivity among matches, which not always hold. In this paper, we present NACluster a non-supervised clustering algorithm for dealing with sky object matching in multiple catalogues. NACluster matching strategy extends the traditional k-means clustering algorithm by relaxing the number k of cluster (i.e. matched sky objects). We experiment NACluster with real and synthetic catalogues and show that the results present better accuracy than state of the art solutions.**

I. Introduction

In astronomy context, catalog is a dataset that contains a list of celestial objects and their characteristics, like position, magnitude and color. There are different astronomical surveys, or catalogues. They are characterized by having different formats, schemes, data structures, since they are handled by distinct and independent projects. Each one covers a specific area of the sky or even the whole area of the Universe. Then, the surveys produce catalogues with intersections in the covered area of the sky. So, It's possible to have objects in common in different catalogues. Therefore, if we want to have an integrated view provided by different catalogues, it's necessary to do the data cross-matching.

Current astronomy surveys present important challenges in the cross-matching area, where the spatial position of objects is very important. The matching tries to identify sky objects registered in different catalogues with slightly different properties but representing the same real object, once there slightly difference in the object position captured by two different telescopes. It can produce ambiguity in the matching. The cross-matching among catalogues is usually applied in peer-to-peer fashion, between two different catalogues, and generates a single output catalog identifying common objects between surveys. The algorithm selects matches considering the shortest distance between objects using a spatial radius ϵ defined by the user. However, when we want to compute a matching among three or more catalogues, a more careful process must be applied, as one shall not consider matching transitively and the ordering with which catalogues are chosen may produce different results.

Match transitivity problem occurs, for example, when given three objects O_1, O_2 and O_3 from different catalogues, the O_2 match with O_1 and O_3, but O_1 does not match with O_3. Thus, $O_1 = O_2$, $O_2 = O_3$, but $O_1 \neq O_3$. In this situation, we would expect that $O_1 = O_2 = O_3$.

Few works in the literature tackled cross-matching in astronomical research domain. Particularly, in [1] some cross-matching algorithms for astronomic catalogues were evaluated. In this work, Q3C Join algorithm [3] was chosen to be evaluated. However, Q3C generated some incorrect matching in the order of billions of false positives when using big catalogues. This problem is due to matching strategy of Q3C, which is ambiguity preserving. In fact, an ambiguity resolution solution is required in this context, which motivated this work.

In this paper, we propose the *NACluster*, a non-supervised clustering algorithm for matching multi catalogues. The aim of this algorithm is to split into clusters the celestial objects present in *n* catalogues, such that in each cluster there are only objects from different catalogues and representing the same real object. This contribution allows the improvement on state of the art astronomy catalog matching.

The rest of the paper is organized as follows. In Section 2, we propose the NACluster algorithm. In Section 3, we present the experiments using this algorithm and its validation, as well as a comparison with the Q3C Join algorithm [3]. Finally, section 4 concludes and presents the future works.

II. NACluster: A Non-supervised Clustering algorithm

The **NACluster** algorithm is short for the **N-way Astronomical Clustering** algorithm, a non-supervised clustering algorithm for matching multi catalogues. The algorithm takes as input n catalogues and produces a clustering composed of k clusters. In defining the matching criteria among objects, an important restriction is that all objects falling in a cluster shall originally come from different catalogues. Furthermore, each

cluster is supposed to represent a real object in the observed sky. Thus, one doesn't know in advance how many objects will be produced by the merged catalogues. This restrictions refrain us from adopting a traditional clustering algorithm, such as k-means.

The pseudocode of the NACluster algorithm is described in Algorithm 1.

The algorithm receives as input a set of catalogues. Each catalog is a set of tuples of form (id, ra, dec), where id is the object identification, ra and dec represents the spatial coordinates of this object. In line 3 of Algorithm 1, the largest catalog is selected and one cluster is created for each its object. In this step, the location of each object is taken as a cluster centroid.

The idea of the algorithm is to compare each object of one catalog to all computed cluster centroids, by computing the distance $d(O_i, C_a)$ of an object O_i to a centroid C_a . When $d(O_i, C_a) < \epsilon$, then the object O_i is mapped to cluster C_a. This mapping, however, can only be applied when there not exists another object O_j in cluster C_a that has been mapped to the same catalog of O_i. In case an object of C_a already exists in the cluster, two scenarios must be evaluated: (1) if $d(O_j, C_a) > d(O_i, C_a)$ then we should remove the object O_j from the cluster C_a, insert O_i in this cluster, and search another cluster for O_j; (2) if $d(O_j, C_a) < d(O_i, C_a)$ then the algorithm performs a recursive search on the candidate list ($candidates$), using the procedure *searchCentroid* (detailed in Algorithm 2), for allocating O_i.In case, no cluster is found at distance ϵ then a new cluster C_b is created to the point O_i and it will be the centroid.

Once all objects from the merged catalogues have been evaluated, a new iteration is performed searching for better cluster centroids. The algorithm finishes when the centroids are stable, i.e. all the computed centroids of an iteration are the same as the previous iteration.

Algorithm 1 The NACluster algorithm

```
1: Input: Dataset containing a set of catalogues
2: procedure CLUSTERING(Dataset)
3:     initializeClusters(largestCatalog);
4:     visitedCatalogs.add(largestCatalog);
5:     while not centroids stable do
6:         if iterationNumber > 1 then
7:             Clear the clusters, but keep their centroids;
8:         end if
9:         for all cat ∈ Dataset & cat ∉ visitedCatalogs do
10:            visitedCatalogues.add(cat);
11:            for all obj ∈ cat do
12:                minDistace ← Double.MAX_VALUE;
13:                existCentroid ← false;
14:                for all centroid ∈ centroids do
15:                    distance ← computeDistance(obj,centroid);
16:                    if distance < ϵ then
17:                        existCentroid ← true;
18:                        candidates.add(centroid,distance);
19:                        if minDistance ≥ distance then
20:                            minCentroid ← centroid;
21:                            minDistance ← distance;
22:                        end if
23:                    end if
24:                end for
25:                if existCentroid then
26:                    if o ∉ c[minCentroid] | o.cat=obj.cat then
27:                        c[minCentroid].add(obj);   ▷ c is a clusters list
28:                        computeCentroid(c[minCentroid]);
29:                    else
30:                        searchCentroid(obj,candidates);
31:                        candidates.clear();
32:                    end if
33:                else
34:                    create cluster;
35:                    cluster.add(obj);
36:                    computeCentroid(cluster);
37:                end if
38:            end for
39:        end for
40:    end while
41: end procedure
42: Output: All clusters and their objects and centroid;
```

III. EXPERIMENTS

In order to evaluate the NACluster algorithm we selected a dense part of 2MASS [6] catalog (90 thousand objects relatively close together) and we applied a normal distribution function for generating new catalogues from 2MASS, simulating in this manner real variations of the same catalog. We have synthesized 5 different datasets containg from 2 to 6 catalogues using this approach.

We evaluate the NACluster by assessing the "quality" of the clustered objects produced by the algorithm. In particular, we use precision, recall, and F-measure having as baseline objects generated from the real objects. Our goal is to evaluate the performance of the NACluster in identifying this set of matched objects. We remind the reader that precision is the fraction of the set of objects retrieved that are relevant; recall is the fraction of relevant objects that were retrieved; F-measure is the harmonic mean between precision and recall [5].

Table I presents the precision, recall and f-measure of the results generated by NACluster. An interesting observation concerning this table is that f-measure remains around 0.97, even the number of merged catalogues in the dataset is increased.

TABLE I. THE EFFECT ON THE PRECISION AND RECALL OF THE NACLUSTER ALGORITHM USING THE 2MASS CATALOG MATCHED AGAINST 1 TO 5 SYNTHETIC VERSIONS GENERATED FROM IT BY APPLYING A NORMAL DISTRIBUTION FUNCTION.

No. of catalogues	Precision	Recall	F-Measure
2	0.9750	0.9763	0.9757
3	0.9717	0.9727	0.9722
4	0.9654	0.9671	0.9662
5	0.9699	0.9713	0.9706
6	0.9734	0.9745	0.9739

TABLE II. THE EFFECT ON THE PRECISION AND RECALL OF NACLUSTER ALGORITHM WHEN MATCHING, IN A SPARSE REGION OF THE SKY, THE UBVRI CATALOG AGAINST 1 TO 5 SYNTHETIC GENERATED VERSIONS OF IT.

No. of catalogues	Precision	Recall	F-Measure
2	0.9937	0.9938	0.9938
3	0.9944	0.9944	0.9941
4	0.9954	0.9960	0.9957
5	0.9932	0.9932	0.9926
6	0.9913	0.9914	0.9906

Algorithm 2 The recursive procedure to search the appropriate cluster $cluster$ with centroid $centroid$ to the object obj from the catalog $obj.cat$ in the centroids candidates list $candidates$

```
1: Parameters: the object and the candidates list
2: procedure SEARCHCENTROIDE(obj,candidates)
3:     if candidates.size = 0 then
4:         create cluster;
5:         cluster.add(obj);
6:         computeCentroid(cluster);
7:     else
8:         minDistance ← Double.MAX_VALUE;
9:         for all centroid ∈ candidates do
10:            distance ← candidates.getDistance(centroid);
11:            if distance < minDistance then
12:                minCentroid ← centroid
13:                minDistance ← distance
14:            end if
15:        end for
16:        if o ∉ c[minCentroid] | o.cat=obj.cat then
17:            c[minCentroid].add(obj);            ▷ c is a clusters list
18:            computeCentroid(c[minCentroid]);
19:        else
20:            oldObj ← o;
21:            oldDistance ←computeDistance(oldObj,centroid);
22:            if minDistance < oldDistance then
23:                c[minCentroid].remove(oldObj);
24:                c[minCentroid].add(obj);
25:                computeCentroid(c[minCentroid]);
26:                candidates.clear();
27:                minDistance ← Double.MAX_VALUE;
28:                for all centroid ∈ centroids do
29:                    distance ← computeDistance(OldObj,centroid)
30:                    if distance < ϵ then
31:                        if centroid ≠ minCentroid then
32:                            candidates.add(centroid,distance);
33:                        end if
34:                    end if
35:                end for
36:                searchCentroide(oldObj,candidates);
37:            else
38:                candidates.remove(minCentroid);
39:                searchCentroid(obj,candidates)
40:            end if
41:        end if
42:    end if
43: end procedure
```

Another similar experiment was done using the UBVRI catalog [2], which is a sparse catalog containing 49,167 objects. In this case precision, recall and f-measure is around 0.99 (refer to Table II), keeping the same accuracy even when the number of merged catalogues are increased.

We can conclude from the previous results that our algorithm has a good accuracy rate, and this rate varies according to the catalog density. In all case, our algorithm converged within two iterations.

A. Comparison with Q3C Join

The *Q3C Join* [4], as most cross-matching algorithms, performs the matching of only two catalogues at a time, producing, as output, a catalog containing the matched objects. We compare the output of the *Q3C Join* algorithm execution with the output of the clustering algorithm *NACluster* taking as input the same two catalogues, both performing the matching between part of the *2MASS* catalog (1 million objects) and the synthetic catalog generated from it (1 million objects).

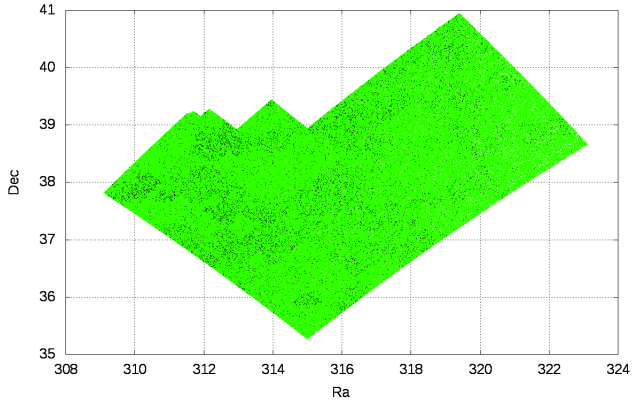

Fig. 1. The colored area represents the region of space occupied by the objects present in part of the 2MASS catalog used in experiments with the NACluster algorithm. Each light green point represents an object correctly classified, i.e., it formed cluster with its real pair. Each black point is an object wrongly classified, i.e. it didn't form cluster with its real pair.

The NACluster algorithm produced as output 1,001,005 clusters, 980,282 of which were correct and 20,723 were wrong, with a hit rate around 98% and 2% error. The Figure 1 shows the region of the sky covered by the catalogues used in the comparison. The x axis is the Right Ascension coordinate, or Ra, and the y axis is the Declination, or Dec. Each light green point is a correctly classified object, i.e. its cluster was formed with its original pair. It is important to mention that the correctly classified points were plotted before the wrongly classified points,as due to the low resolution of the plot, the points take up more space than their actual size. Thus, we can have a better view of the comparison.

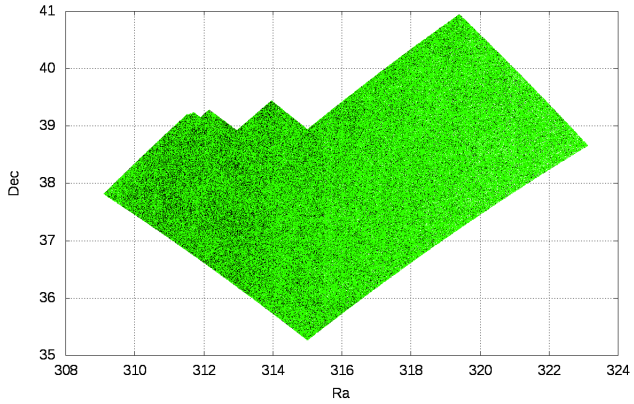

Fig. 2. The colored area represents the region of space occupied by the objects present in part of the 2MASS catalog used in experiments with the Q3C Join (radius 0.001 degree was used as parameter). Each light green point is an object classified correctly, i.e., it formed cluster with its real pair. Each black point is an object wrongly classified, i.e. it didn't form cluster with its real pair.

The Q3C Join presented an output dependent on the size of the radius used. Figures 2 and 3 illustrate the case, they follow the same caption of Figure 1. Figure 2 shows the result of the matching using a radius 0.001 degree. This radius allowed to retrieve 964,432 correct matches, but also retrieved 48,624 wrong matches. One can see that the black area of Figure 2 is much larger than in Figure 1.

We changed the radius to 0.002 degree in order to achieve all synthetic objects and understand the behavior of Q3C Join. The result is shown in Figure 3, in which the black area occupies the large area of the graphic, because the output of Q3C Join using the radius 0.002 degree returned 339,536 wrong matches, ie, 33.95% of matches, apart from the 1 million correct matches, ie all the real matches.

Table III presents the precision, recall and F-measure for these same experiments. Values in bold correspond to the highest precision, recall and F-measure. Note that the scenario using NACluster outperforms all other scenarios in precision and F-measure. When the radius in Q3C Join is 0.002 degree, its recall (1.0) is higher than the recall (0.9875) shown in the proposed approach, but its precision (0.7456) is considerably worse, leading to a relatively low F-measure (0.8549).

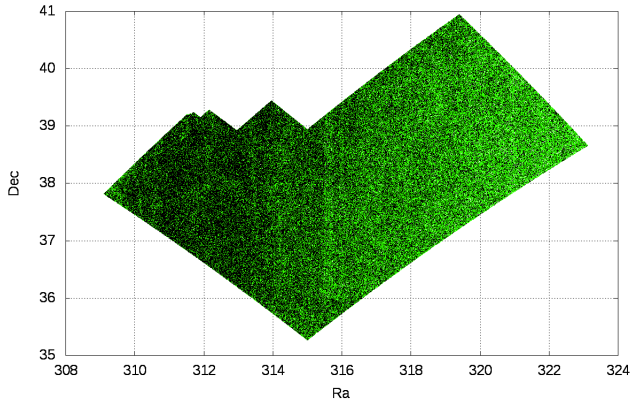

Fig. 3. The colored area represents the region of space occupied by the objects present in part of the 2MASS catalog used in experiments with the Q3C Join (radius 0.002 degree was used as parameter). Each light green point is an object classified correctly, i.e., it formed cluster with your real pair. Each black object is an object classified wrongly, i.e. it not formed cluster with your real pair.

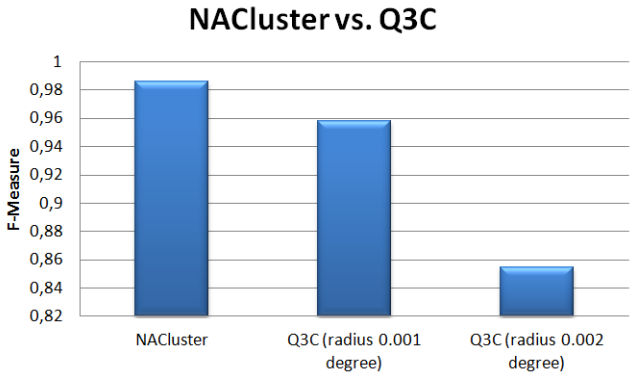

Fig. 4. Comparison among NACluster and Q3C Join

Finally, Figure 4 shows a direct comparison of the F-measure for these scenarios. According that, NACluster algorithm has performance superior than the Q3C Join, even increasing its radius.

The result of this comparison leads us to conclude that, unlike the strategy of the NACluster algorithm to solve the matching ambiguity, the Q3C Join has a strategy to preserve the ambiguity, accepting duplicity at matches. This duplicity was also identified in the experiments presented in a previous paper [1].

TABLE III. COMPARATIVE RESULTS, IN TERMS OF PRECISION AND RECALL, CONFRONTING MATCHING STRATEGIES BASED ON Q3C AND NACLUSTER, USING THE 2MASS AND A SYNTHETIC GENERATED CATALOG DERIVED FROM IT.

Algorithm	Precision	Recall	F-measure
NACluster	**0.9834**	0.9875	**0.9857**
Q3C (0.001 degree)	0.9520	0.9644	0.9582
Q3C (0.002 degree)	0.7456	**1.0**	0.8549

IV. CONCLUSION AND FUTURE WORKS

In this paper, we presented the NACluster algorithm, a non-supervised clustering for matching multiple catalogues.

In order to evaluate the effectiveness of this algorithm, we used real astronomical catalogues, which were useful to generate new catalogues with perturbations on objects locations by using a normal distribution. These preliminary results indicate that the algorithm is effective in matching objects from different catalogues. Thus, this contribution allows improving the state of the art astronomy catalog matching.

The algorithm presents exponential complexity on the number of individual sky elements (i.e. clusters). By using a spatial indexing strategy, the actual number of comparison is reduced to a local region, making the algorithm in practice feasible and presenting a huge potential for parallelization.

In future work, we are planning to develop a parallel strategy for NACluster algorithm, and to evaluate other similarity functions. We intend to use this approach as the first step on entity resolution problems for very large catalogues.

V. ACKNOWLEDGEMENTS

The authors would like to show their gratitude to the Inter-institutional e-Laboratory for Astronomy (LIneA[1]) for having first presented the problem to us and provided the datasets and infra-structure for running the experiments. This work has been partially supported by the FAPERJ-INRIA MUSIC project E-26/110.074/2014.

REFERENCES

[1] V. P. Freire, A. M. d. C. Moura, F. Porto, and J. A. F. de Macêdo, in *BreSci - VII Brazilian e-Science workshop. Proceedings of the XXXIII Congress of the Brazilian Computer Society (CSBC 2013)*, pp. 1870–1875.

[2] A. Henden, "UBVRI measurements of stars by Arne Henden (USNO)," http://stupendous.rit.edu/tass/catalogs/ubvri.tass, 1998.

[3] S. Koposov and O. Bartunov, "Q3C , Quad Tree Cube – The new Sky-indexing Concept for Huge Astronomical Catalogues and its Realization for Main Astronomical Queries (Cone Search and Xmatch) in Open Source Database PostgreSQL," *The Astronomical Data Analysis Software and Systems (ADASS) conference*, vol. 351, pp. 735–738, 2006.

[4] S. Koposov and O. Bartunov., "Q3C," http://code.google.com/p/q3c/, 2012.

[5] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, Jul. 2008.

[6] M. Skrutskie, R. M. Cutri, and et al., "The Two Micron All Sky Survey (2MASS)," *AJ*, vol. 131, no. 2, pp. 1163–1183, 2006.

[1] http://www.linea.gov.br