**Distance Matching algorithm:**

1. **Pre-computing the distances of every sky object from other objects in the dataset in a specific boundary:**

We pre-compute the distances of every sky object from all other objects in a specific boundary of it (we can use quadtrees or octrees for this purpose) and store an ordered list of **m** - least distances of **m**-nearest points of the correspondent sky object. By doing this we are able to answer the future pattern queries with maximum number of **m**-points. Here is the result of this step:

|  |  |  |  |
| --- | --- | --- | --- |
| **Astronomy dataset (the original catalogue + 2 new columns)** | | | |
| **star-id** | **…** | **m-partners**  **(in ascending distance order)** | **partners-distances**  **(ascending order)** |
| **…** | **…** |  |  |
| **s12** | **s15, s18, s19, s21, s33, s35, …, sm** | **d12\_d15,…,d12\_m** |
| **s13** | **…** | **…** |
| **…** | **…** | **…** |
| **s46** | **s41, s42, …** | **d46\_41, d46\_42, …** |
| **…** | **…** | **…** |
| **s67** | **s15, s65, …** | **d67\_15, d67\_65, …** |
| **…** | **…** | **…** |

1. **Pre-compute the Query:**

* Identify some small number of points k' among q1 to qk that are far apart from one another. We want those because the error matters less when points are far apart. Then, we perform the join-style algorithm on the sets corresponding to those k' points and then for each candidate group of stars that survive that filter, test all possible pairs.
* Find the more representative points in the query (should be discussed more).

1. **Calculate the pairwise query points distances:**

For example in a query with k = 4 points, we compute all **k(k-1)/2** pairwise distances and then choosing the centroid(here q0) we sort the distances in ascending order as following:

**q1**

d0

**q0**

d4

d1

d5

d3

**q2**

d2

**q3**

**q0\_partners (q1, q3, q2)** (in ascending distance order)

**q0\_distances (d0, d3, d4)**

We need the other pairwise distances as well, so:

**Dist[q1][q2] = d1**

**Dist[q1][q3] = d5**

**Dist[q2][q3] = d2**

1. **Find the candidate matches:**

We look for candidate matches with respect to the query centroid: In general, the qi candidates should be sets, so the qi candidates for s12 should be maybe s15, s22, s24

|  |  |  |  |
| --- | --- | --- | --- |
| centroid candidate  (star-id) | **i** | **qi candidate**  (star-id) | qi position |
| **s12** | **1** | **s15** | s15(ra-value, dec-value) |
| **s35** | **1** | **s18** | s18(ra, dec) |
| **s46** | **2** | **s18** | s18(ra, dec) |
| **…** | **…** | **…** | **…** |

1. **Build the candidate solutions**

We do the join operations in the above table to find the candidate solutions: Please write out the joins explicitly.

<s12, s15, s19, s33>,

<s12, s18, s19, s33>,

<s12, s18, s21, s35>,

<s46, s41, s42, s56>,

…

1. **Check the pairwise distances: This method is correct if there are no errors In position, but the joins I was talking about have to do with handling the errors in position.**

*for each c in C (for each candidate solution)  
  form a 1-1 correspondence between the stars in c and the k+1 query points based on the*

*distances from point0, thus renaming the stars in c point0, point1, .... pointk consistently.*

*For every pair pointu and pointv in point1, … pointk*

*See whether dist(pointu, pointv) corresponds to dist(qu, qv) where qu is the uth point*

*farthest away from q0 and qv is the vth point farthest away from q0*

*If false, then c is not a match  
  c is a match*

1. **Ranking the solutions:**

We rank ascending the solutions by the summation of their pairwise distances: I don’t understand why this is helpful, unless by pairwise distances you mean the sum of the errors in the distances.

<s12, s15, s19, s33>, Total-cost = 21.5

<s12, s18, s19, s33>, Total- cost = 33

<s2, s18, s21, s35>, Total- cost = 33.5

So, 1 through 4 are basically fine except that the Ci with respect to each centroid should be all those stars that are approximately (i.e. within error bounds) dist(q0, qi) from the candidate centroid c0. Let me give you a detailed example. I will consider 1 dimensional space to make life super easy.

q0 should be 2 away from q1, 3 away from q2, 4 away from q3.

q1 should be one away from q2 and 2 away from q3.

q2 should be 1 away from q4

That’s the query. Now consider some stars at locations

r0 = 10, 20

r1 = 10, 24

r2 = 10, 22

r3 = 10, 18

r4 = 10, 17.9

r5 = 10, 16.9

r6 = 10, 16

r7 = 10,23.1

Let’s consider r0 as a potential centroid. Then C1 = {r2,r3,r4} – stars that are 2 away (within an error bound of 0.1). Then C2 = {r5,r7}. C3 = {r6,r1}.

Now when we start doing the pairwise comparisons among C1, C2, C3, we will be doing a bunch of join computations (what I’m asking you to work out with Fabio) that will result in matching quadruples of stars ordered to match q0, q1, q2, q3:

(r0, r2, r7, r1), (r0, r3, r5, r6), (r0, r4, r5, r6). Those are the joins I’m hoping to get from you and Fabio based for example on the table design that I suggested the other day.