**Distance Match:**

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

We pre-compute the distances of every sky object from all other objects 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: As I’ve said before, we can use quadtrees or octrees for this purpose. That will eliminate much computation.

|  |  |  |  |
| --- | --- | --- | --- |
| **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. **Compute 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 one of the points (here q0), Usually we want to choose the centroid we sort the distances in ascending order as following:

d0

**q0**

**q1**

d4

d1

**q2**

d3

d5

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 solutions:**

(I used the notation of **Candidate solutions** that means the list of possible solutions for our pattern query; every candidate solution contains at least one possible candidate match for every query point. The matching degree can be measured by a distance-metric and the match-cost differs from 0 to 1)

We look for candidate solutions like [**q0\_partners (q1, q3, q2)**] with similar distances as [**q0-distances (d0, d3, d4)**] in the **Astronomy-dataset**. Here, we have the results: Yes!

|  |  |  |
| --- | --- | --- |
| q0 candidate  (star-id) | **candidate partners sets** (in ascending distance order)  <<q1 candidates >,<q3 candidates>,<q2 candidates>> | Correspondent distances |
| **s12** | **< <s15, s18> , <s18, s19, s21>, <s33, s35> >** | <<d12\_15, d12\_18>,  <d12\_18,d12\_19,d12\_21>,<d12\_33,d12\_35>> |
| **s35** | **<<s54>, <s70, s74, s71, s45>, <s45, s11, s89>>** | <<d35\_54>,  <d35\_70,d35\_74,d35\_71, d35\_45>,  <d35\_45,d35\_11,d35\_89>> |
| **s46** | **<<s41>, <s42, s21>, <s56>>** | <<d46\_41>, <d46\_42,d46\_21>,  <d46\_56>> |
| **…** | **…** | **…** |

1. Finally, we find the solutions by the following query:

*for each c in C (for each group of candidate stars)  
  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*

Here are the final solutions:

**<s12, s15, s19, s33>,** Total-Match-cost = Tc1

**<s12, s18, s19, s33>,** Total-Match-cost = Tc2

**<s12, s18, s21, s35>,** Total-Match-cost = Tc3

**<s46, s41, s42, s56>,** Total-Match-cost = Tc4

And we cannot have a solution like this:

<s12, s54, s21, s56>

Right, but because we have sets, we can use joins.

Here is how: imagine a table

R(firstqueryelement, secondqueryelement, firstdatastar, seconddatastar)

q0, q1, s12, s15

q0, q1, s12, s18

q0, q3, s12, s18

q1, q3, s15, s18

Etc.

Then you work out the joins. Thanks, Dennis