Subject: Re: yet another 2d matching question
Posted by pgrigis on Fri, 30 Jul 2010 15:59:35 GMT
View Forum Message <> Reply to Message

On Jul 30, 11:41 am, Gray <grayliketheco...@gmail.com> wrote:
> On Jul 30, 11:25 am, Gray <grayliketheco...@gmail.com> wrote:
>
>
>
>> On Jul 30, 11:23 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>> On Jul 30, 11:15 am, Paolo <pgri...@gmail.com> wrote:
>
>>>> On Jul 30, 10:01 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> > Hi all,
>
>>>> > For quite a while I've been using JD Smith's match_2d routine to match
>>>> > xy coords between lists. However, this and all the other matching
>>>> > codes I've seen out there suffer from a variation of the uniqueness of
>>>> > matches problem.
>
>>>> > Codes like SRCOR in the NASA IDL library let you specify a one-to-one
>>>> > match, i.e. enforcing that each element in list 2 only be matched to
>>>> > one element in list 1; using match_2d's match_distance keyword one
>>>> > could implement the same effect oneself. However, while that excludes
>>>> > multiple matches to the same element, it's all done after the fact,
>>>> > after the original match was determined.
>
>>>> > What I'm looking for is an algorithm that matches 2 lists, identifies
>>>> > multiple-matches, and then looks for additional matches within the
>>>> > search radius for elements which would become unmatched after
>>>> > enforcing a one-to-one relationship. What I mean is, say element 0 in
>>>> > list 2 is matched to both element 3 and element 5 in list 1, and that
>>>> > the distance between 2_0 and 1_3 is smaller than the distance between
>>>> > 2_0 and 1_5. In that case, 1_5 would become unmatched; but what if
>>>> > there is element 2_1 which is also within the search radius of 1_5?
>>>> > Then, 1_5 should be re-matched with 2_1.
>
>>>> > My best idea thus far is to run match_2d once, identify multiple-
>>>> > matches, keep the matches with minimum distance using match_distance,
>>>> > then iterate with the remaining elements until match_2d returns no
>>>> > matches. Can anyone come up with a better solution?
>
>>>> Hmmm... what about starting with first point (a) in list 1, finding
>>>> the nearest
>>>> point (b) to (a) in list 2, removing (b) from list 2 and repeat for
>>>> all points

>>>> in list 1? [this assumes list 1 and list 2 have the same number of
>>>> elements N,
>>>> which is a necessary condition for a one-to-one matching].
>
>>>> With some smart partitioning of list 1 it will take ~log(N) to find
>>>> the nearest
>>>> point, so we are looking at ~ N log(N) operations...
>
>>>> Ciao,
>>>> Paolo
>
>>>> > --Gray
>
>>> I'm fine with having there be points which don't match at all w/in the
>>> search radius, I'm just looking to force any matches that exist to be
>>> recognized.
>
>>> The straight FOR-loop method is certainly serviceable, but I had hoped
>>> there was a more efficient way to do it... but it's certainly possible
>>> (or even likely) that anything fancier I try to do is LESS efficient.
>
>>> --Gray
>
>> Though I have trouble believing that FOR is the way to go when I have
>> ~50k elements in each list.
>
> AND... there's no guarantee that the first match you find for a given
> element in list 2 is the best one.

what is the "best" match you would like to obtain?

Ciao,
Paolo