Lecture 33:

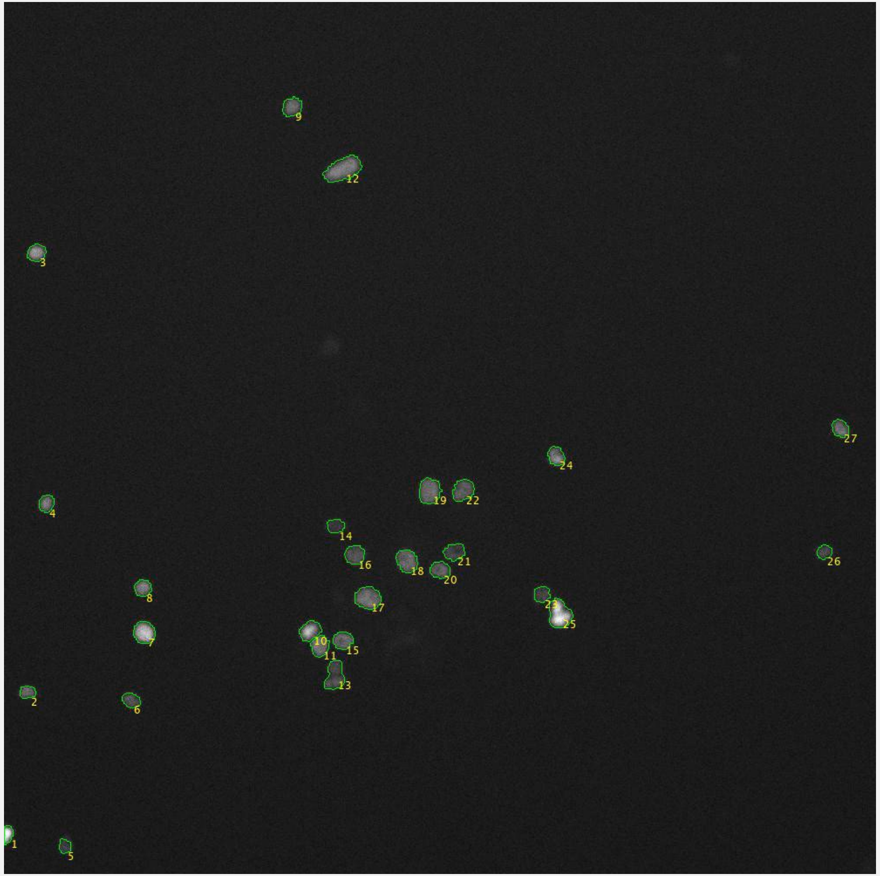
Tracking moving objects

Lecturer: Jian Wei Tay

Date: 12 November 2021

Learning objectives

- Understand how nearest-neighbor tracking works
- Implement tracking using the linear assignment toolbox



Quantitative microscopy can be used to answer questions such as:

- What happens to cells when treated with X

To do this, we need to follow (or <u>track</u>) a cell over time

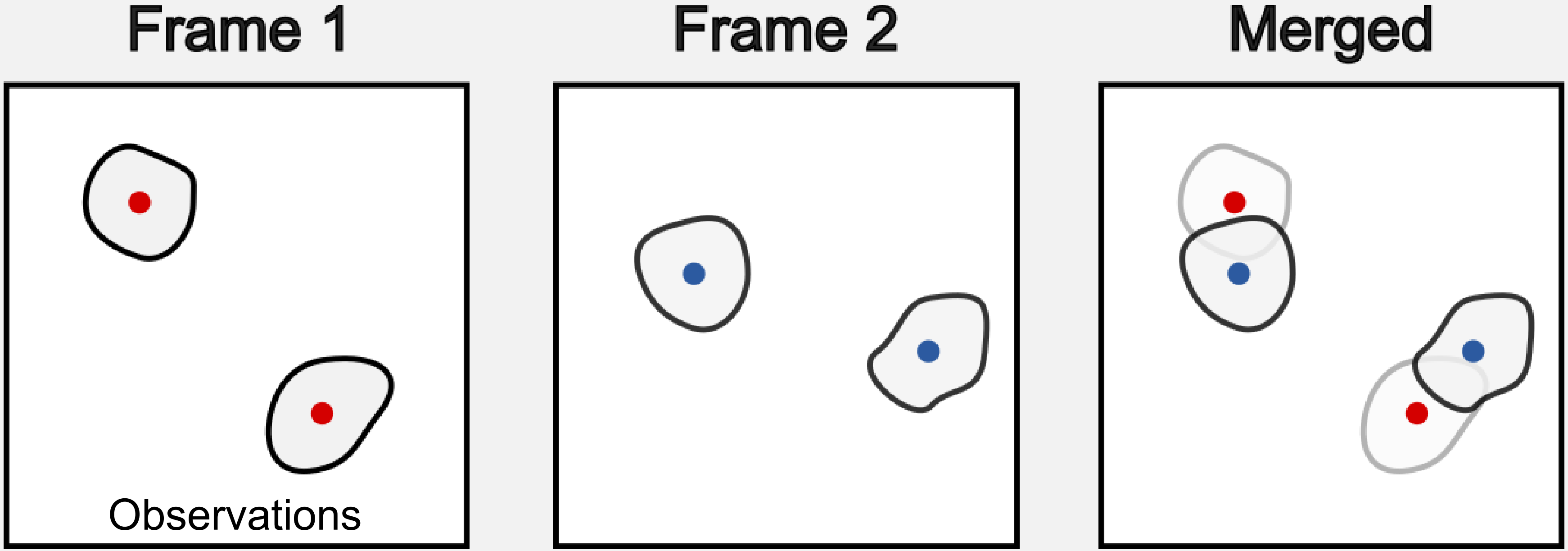
The goals of a tracking algorithm

- Identify individual objects in a movie
- Organize this information into a data structure that allows time-series data to be analyzed

Terminology

- **Linking** – the process of associating objects in one frame with objects in another
- **Observations** – data from objects in current frame (i.e., from regionprops)
- **Track** – a collection of time-series data belonging to a single object (e.g., position, mean intensity over time)

The tracking problem



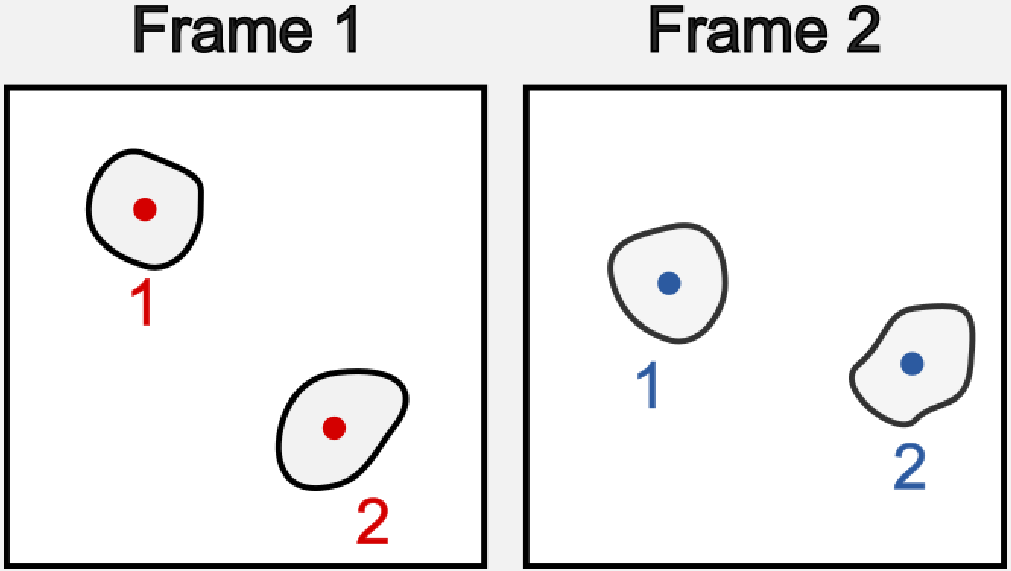
Microscope collects snapshots of the cells. How do we link these objects between frames?

Nearest neighbor algorithm

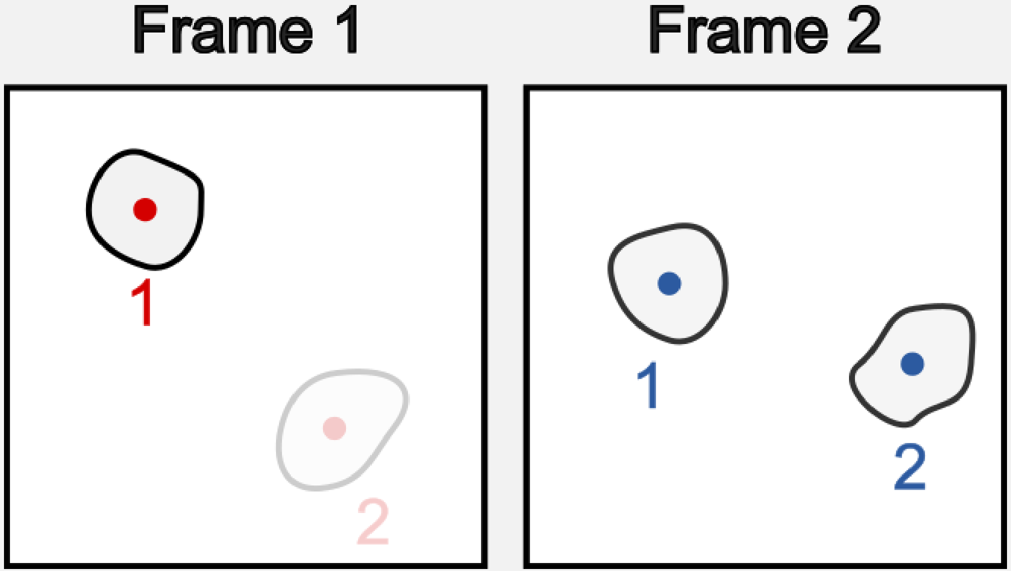
- Assumption: Cells move slowly compared to frame rate of acquisition

- When comparing object positions between frames, a pair of observations with the shortest distance should be linked together

The nearest-neighbor tracking algorithm step-by-step

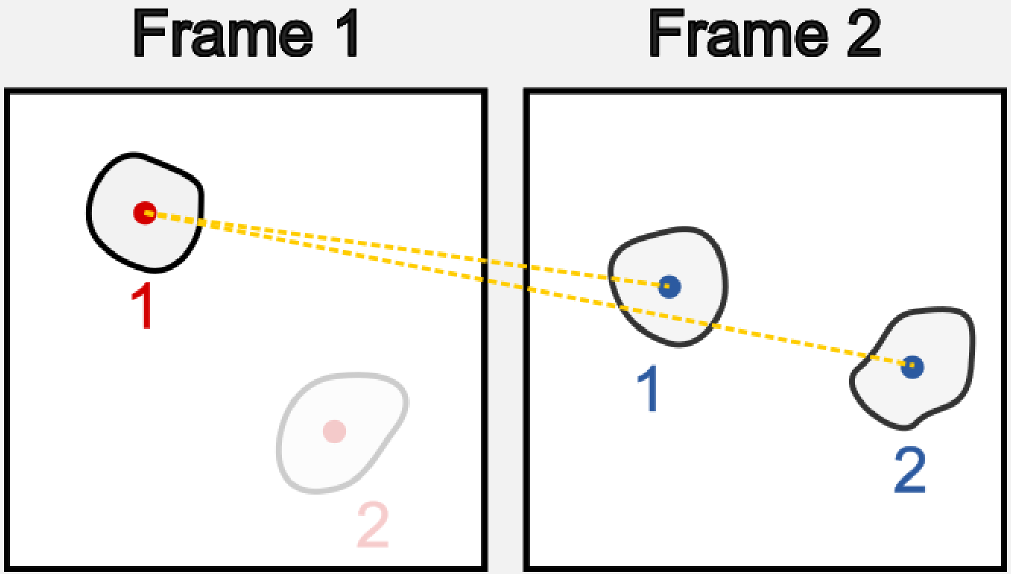


The nearest-neighbor tracking algorithm step-by-step



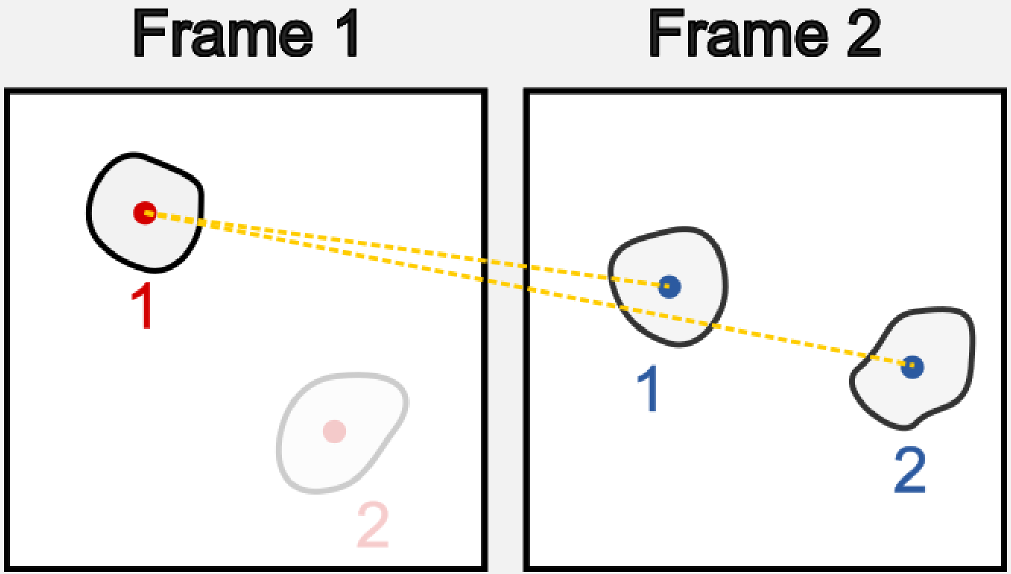
Ignore cell 2 in frame 1 for now

The nearest-neighbor tracking algorithm step-by-step



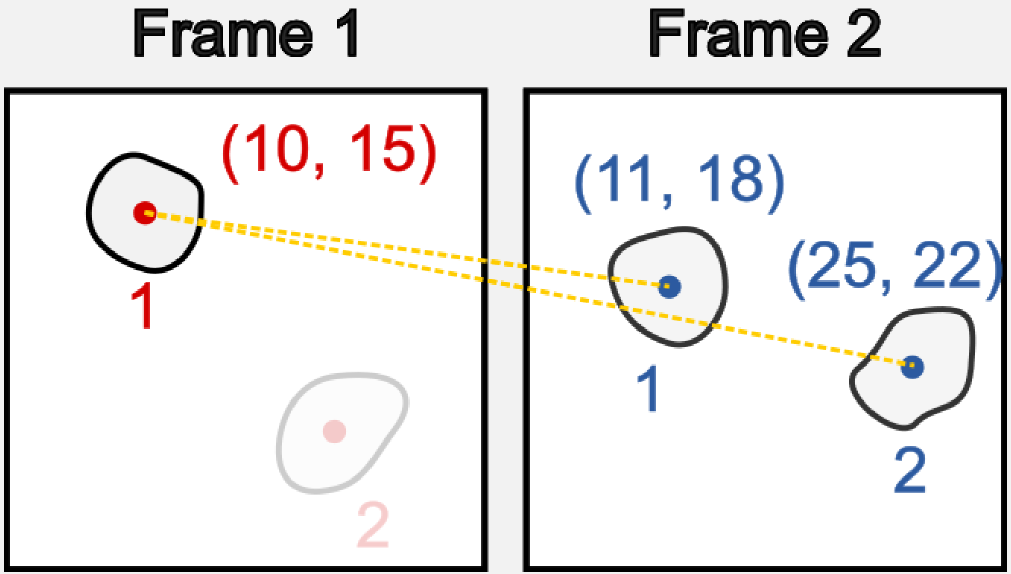
Compute distance between cell 1 in frame 1 with every cell in frame 2

The nearest-neighbor tracking algorithm step-by-step



Compute distance between cell 1 in frame 1 with every cell in frame 2

The nearest-neighbor tracking algorithm step-by-step

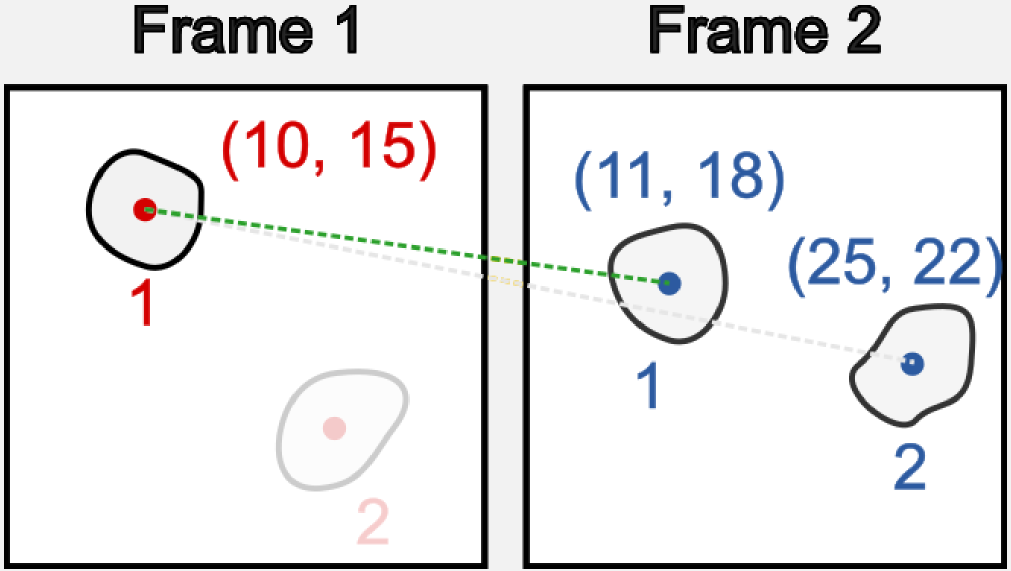


Compute distance between cell 1 in frame 1 with every cell in frame 2

$$d_{11} = \sqrt{(10-11)^2 + (15-18)^2} = 3.16$$

$$d_{12} = \sqrt{(10-25)^2 + (15-22)^2} = 16.6$$

The nearest-neighbor tracking algorithm step-by-step



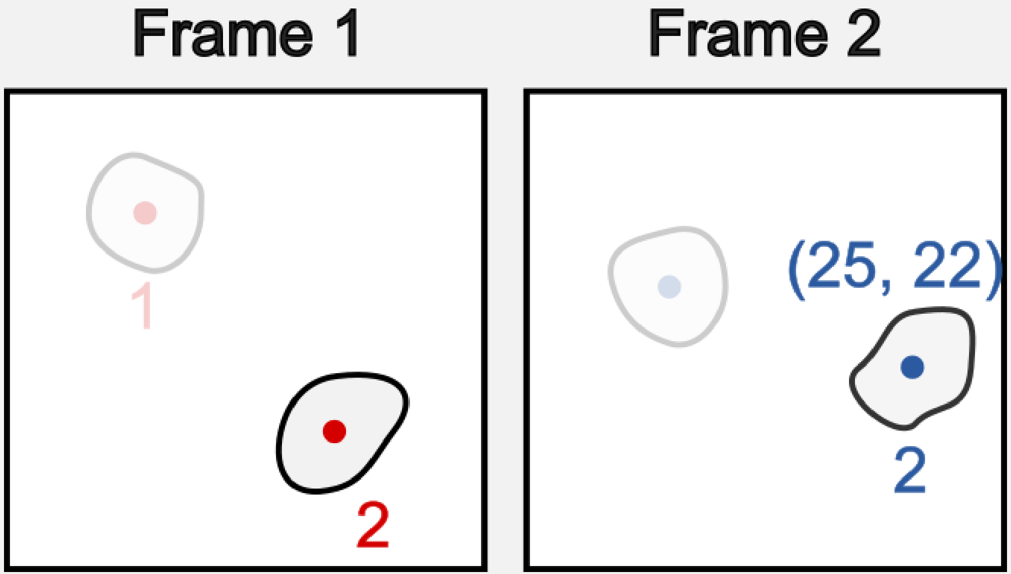
Compute distance between cell 1 in frame 1 with every cell in frame 2

$$d_{11} = \sqrt{(10-11)^2+(15-18)^2}$$
$$= 3.16$$

$$d_{12} = \sqrt{(10-25)^2+(15-22)^2}$$
$$= 16.6$$

Find the shortest distance

The nearest-neighbor tracking algorithm step-by-step

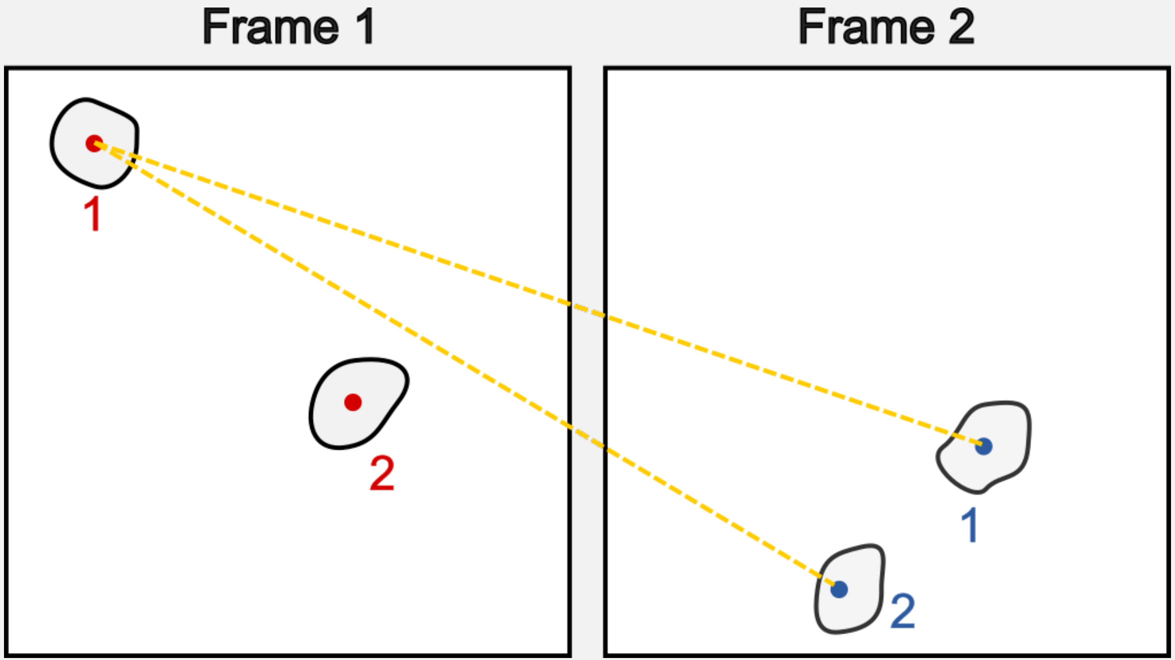


Move to the next cell (cell 2 in frame 1)

But remove linked objects from future consideration

In this example, cell 2 in frame 1 only has a single valid neighbor

Prevent "impossible" distances from being linked



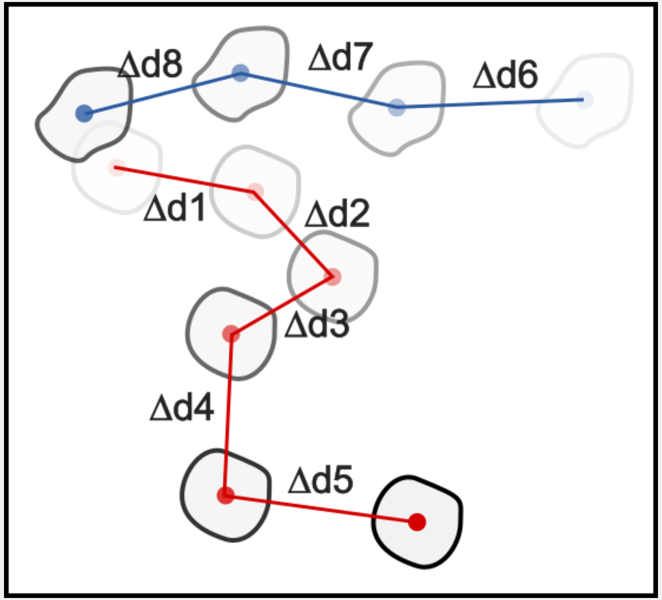
In this example, cell 1 in frame 1 has drifted out of the field of view.

However, if we carry out the nearest-neighbor calculation it would link to cell 1 of Frame 2.

To avoid this, we set a maximum value for this linking distance.

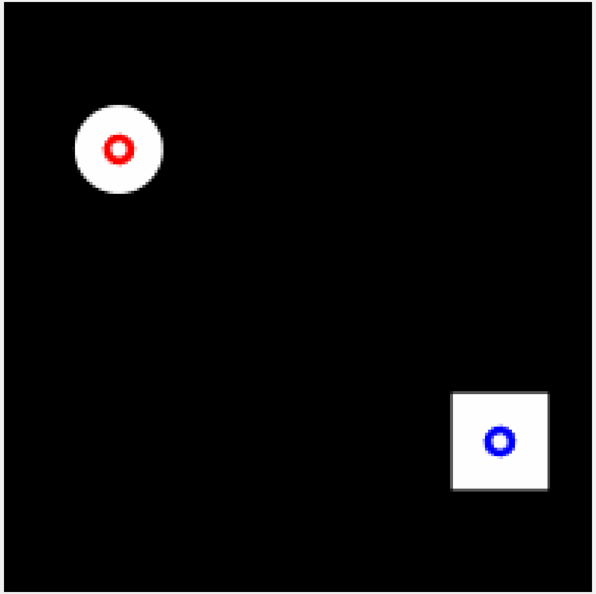
Maximum distance for linking

- The maximum distance for linking should be ~2 – 3x the average step size of objects between frames



Issue with the simple nearest-neighbor tracking

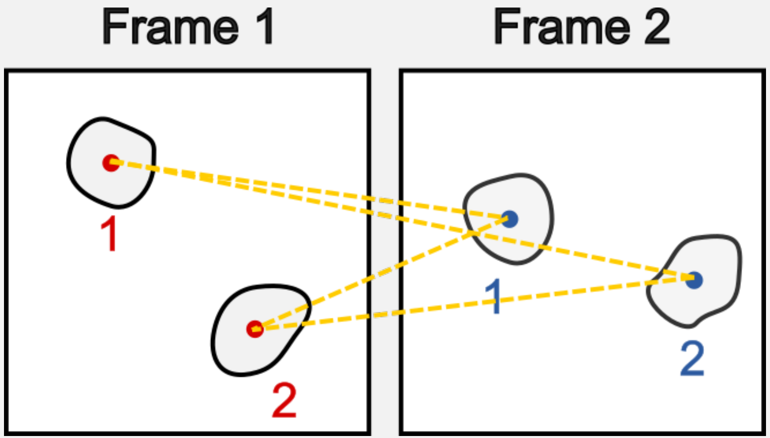
Simple nearest-neighbor makes mistakes when objects "cross paths"



This happens (a lot) when cells are clustered in an image

The linear assignment approach minimizes this problem

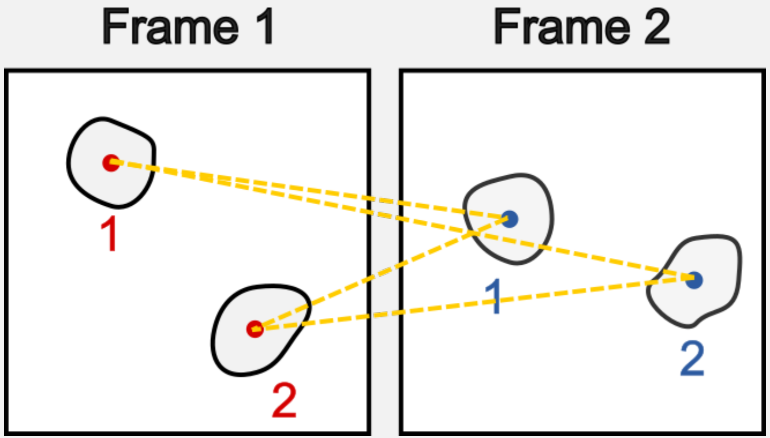
- In the linear assignment approach, the distance of every cell in frame 1 to every cell in frame 2 is computed in a matrix



Frame 1 \ Frame 2	Cell 1	Cell 2
Cell 1	3.16	16.6
Cell 2	14.8	1.24

The linear assignment approach minimizes this problem

- Assignment = choosing a column for each row
- Cells are assigned to <u>minimize the total distance</u>



Frame 1 \ Frame 2	Cell 1	Cell 2
Cell 1	3.16	16.6
Cell 2	14.8	1.24

Nearest-neighbor

If we used the simple nearest neighbor algorithm, then object 1 in frame 1 (the circle) will have distances

$$d_{11} = \sqrt{(232 - 263)^2 + (143 - 327)^2} = 186.6$$

$$d_{12} = \sqrt{(232 - 133)^2 + (143 - 477)^2} = 348.4$$

The algorithm would link object 1 (circle) in frame 1 with object 1 (square) in frame 2.

Then object 2 in frame 1 HAS to link with object 1 in frame 2.

Using the linear assignment approach

Frame 2

Frame 1	Object 1	Object 2
Object 1	186.6	348.4
Object 2	228.4	418.0

Assign columns to rows to minimize the total distance

There are only two possibilities:
1>1, 2>2: Total cost = 186.6+418.0 = 604.6

Using the linear assignment approach

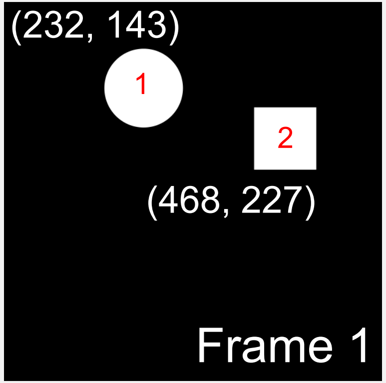
Frame 1 \ Frame 2	Object 1	Object 2
Object 1	186.6	348.4
Object 2	228.4	418.0

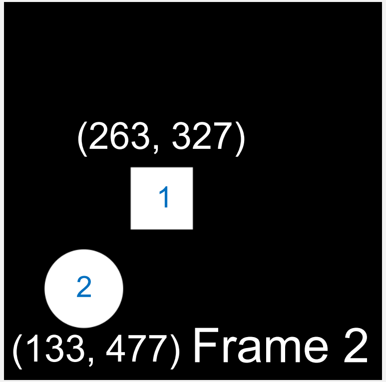
Assign columns to rows to minimize the total distance

There are only two possibilities:
1>1, 2>2: Total cost = 186.6+418.0 = 604.6
1>2, 2>1: Total cost = 228.4+348.4 = 576.8

The total cost is lower even though we assigned object 1 not to i

Using the linear assignment approach





Frame 1 \ Frame 2	Object 1	Object 2
Object 1	186.6	348.4
Object 2	228.4	418.0

The total cost is lower even though we did not assign object 1 to its nearest neighbor

Assign columns to rows to minimize the total distance

There are only two possibilities:

1>1, 2>2: Total cost = 186.6+418.0 = 604.6

1>2, 2>1: Total cost = 228.4+348.4 = 576.8

Summary

- Tracking is an important image analysis problem – we want to follow an object over time to get biologically relevant data

- Objects are linked by distance (usually)

- The linear assignment approach links objects by minimizing the total distance of all links

You can use other properties to link objects

- So far, we've only looked at using distances
- But you can compare sizes, shapes, etc... or even a combination by changing the cost function
- Example: Cost = Δdistance + Δarea

General limitations

- Tracking accuracy will decrease depending on cell density – more dense, less accurate

Experimental considerations

- Initial density of cells
  - Assuming cells are growing, how dense can your movie get while still segmenting and tracking cells accurately?
- Frame rate of movie
  - Movie must be recorded fast enough to meet the assumption that cells do not move "much" for tracking
  - But not so frequently that the laser causes photobleaching or phototoxicity

MATLAB Implementation

The Linear Assignment toolbox

- This is a general purpose toolbox I wrote to implement the linear assignment tracking algorithm
  - https://github.com/Biofrontiers-ALMC/cell-tracking-toolbox/releases

How to use the toolbox

- First create a LAPLinker object

```
>> linker = LAPLinker;
```

Linking options are in the object properties

- The only option you need to change should be `LinkScoreRange`
- This option specifies the range of valid distances for linking objects: `[minDistance, maxDistance]`
- By default, the maximum distance is set to 100 px but you should change this after watching the movie

To change linking score range

- Assign the value to a 1x2 matrix:

Example:

```
>> linker.LinkScoreRange = [0 200];
```

To track data

- The object was written to use the output struct of regionprops

- To get the code to track objects, you only need to use the method `assignToTrack`. The syntax is:

```
linker = assignToTrack(linker, frameNumber, dataStruct)
```

General code structure

```
%Initialize the LAPLinker object
linker = LAPLinker;
linker.LinkScoreRange = [0 250];


for iT = 1:numFrames
    %Read in image
    I = imread('file.tif', iT);

    %Make mask, watershed etc…
    mask = imbinarize(I);

    %Measure data
    data = regionprops(mask, I, 'Centroid', 'MeanIntensity')

    %Track data
    linker = assignToTrack(linker, iT, data)

end
```

Example

- Track cells in the image file `nuclearMask.tif`

- Using the file L33_trackingExample.m, <u>add the three lines</u> of code that are missing to track the objects

- Note: This image contains just the mask of cell nuclei. For your homework, you will need to make the masks yourself.

Saving tracked data

- Once your code has finished running, you can save data using the function save

Example: Save the linker object

```
>> save('trackData.mat', 'linker')
```

To load the data, you can double-click the .mat file to open it in MATLAB

Data from the toolbox

- To get data, use the method `getTrack`

- Example: To get data from track 1

```
t1 = getTrack(linker, 1)
```

Data is stored in a multi-element struct

- Each element of the struct corresponds to an individual object

```
         ID: 1
   MotherID: NaN
 DaughterID: NaN
     Frames: [1 2 3 4 5 6 7 8 9 10]
       Area: [10×1 double]
   Centroid: [10×2 double]
```

Don't worry about these for now – these indicate if cells divide

Accessing time-series data

- Data of each cell (measured using regionprops) is stored as a matrix/vector in each named field of the track structure

- New time-points are added along the rows

- Example: To get the centroid position of the cell in frame 7

```
>> position = T1.Centroid(7, :);
```

Accessing time-series data

- The Frames property shows the frames that this object has been tracked in

- Note: The first frame might not be 1 (e.g., if the cell divided, then the first element of Frames will be the first frame the daughter cell was first detected in)

Plotting time-series data

- To plot data in a vector, you can use the function `plot`

```
plot(xVector, yVector)
```

Example

- This movie was recorded with a frame rate of 10 mins per frame. Plot area vs time in minutes.

Example

- This movie was recorded with a frame rate of 10 mins per frame. Plot area vs time in minutes.

```
timeVec = (Frames) * 10;
plot(timeVec, Area)
xlabel('Time (mins)')
ylabel('Area (pixels)')
```

Note

- Just like last week, I've supplied an example script if you need help getting started

- The script also includes some lines of code to make a video to show the results of the tracking. You may need to install the Computer Vision Toolbox (Add-Ons > Get Add-Ons and search for this toolbox).

- You don't need to use the code – you can write your own from scratch