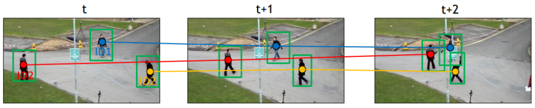
MOT

Friday, May 31, 2024 8:44 AM

Multiple Object Tracking (MOT) task

- Typically assumes a pretrained object detector available (e.g., pedestrians)
- Analyze detections to recover trajectories for "all" specified class of objects

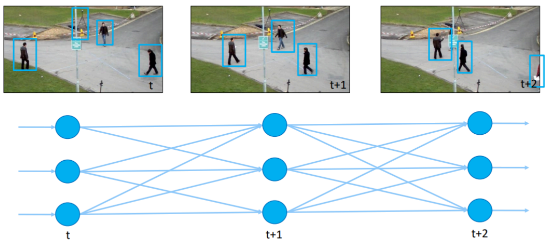


- Assume that objects may:
  (i) enter the scene, (ii) leave the scene or get occluded, (iii) re-enter the scene
- The number of objects (in general) unknown in advance

In many real-world applications of MOT, the exact number of objects to be tracked is not known beforehand. This uncertainty adds complexity to the tracking process, as the system must dynamically adjust to the varying number of objects. Algorithms must be designed to initialize new object tracks as they appear and terminate tracks as objects leave the scene, all while maintaining high accuracy and performance.

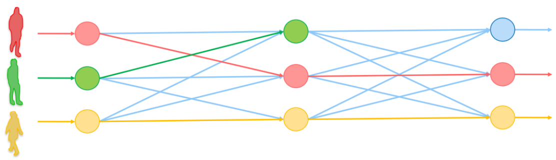
Batch tracking – Network Flow formulation

- Tracking: finding paths through a graph constructed over a sequence of detections *(detection=node, between-frame track=vertex)*



Network flow formulation

- Solve the Minumum Cost Flow problem:
  "Determine the minimum cost of shipment of a commodity through a network"



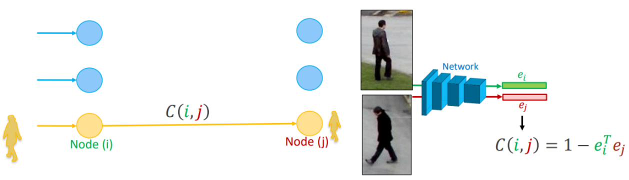
Node = detection ; Edge = flow = trajectory ; 1 unit of flow = target

Find a set of trajectories that minimize the flow cost

To each edge (I, j) we assign a cost **C(I, j)** and an activation indicator:

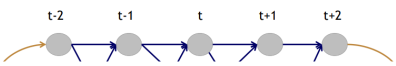
- $f(i,j) \in \{0,1\}$

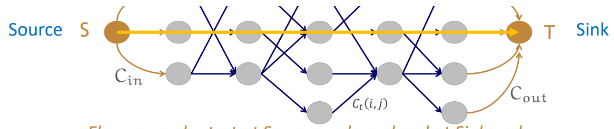
The cost may be a similarity function between two detection (for example a dot product):



- High similarity = low cost -> $C(i,j)$ ↓
- Low similarity = high cost -> $C(i,j)$ ↑

The network can then be expressed as:





Flow can only start at Source node and end at Sink node.

- Transition cost $C_t(i,j)$: appearance difference between detections
- Entrance/exit $C_{in}(i)/C_{out}(i)$: cost to start and end a trajectory

$$\mathcal{T}* = \arg\min_{\mathcal{T}} \sum_{i,j} C(i,j)f(i,j)$$

BUT, since all costs are >0, the trivial solution is $f(i,j) = 0$, for all (i,j)!

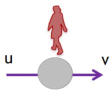
To slove for trivial solution, we introduce a negative cost that reflects the quality of the detections:

Introduce a negative cost that reflects the quality of the detection

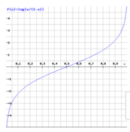
$\beta_i = 1 - \alpha_i$ ... probability that a detection *(i)* is a false alarm

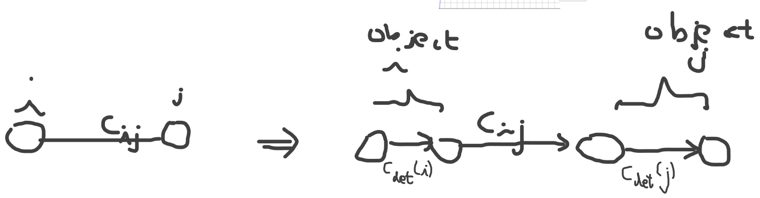
We then augment the graph by a "**detection edge**":

- Augment the graph by a "detection edge" for each node



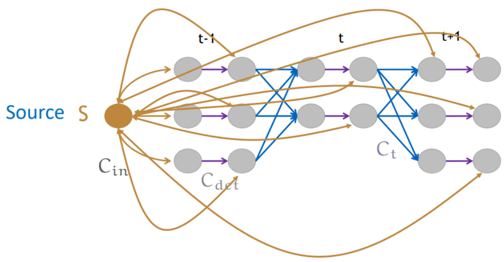
$$C_{det} = \log \frac{\beta_i}{1-\beta_i}$$





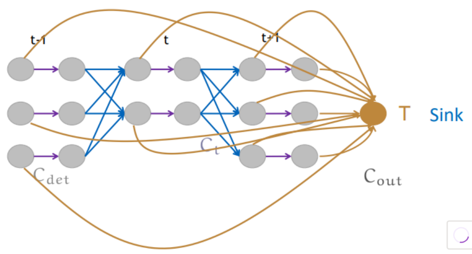
To allow for new detections to happen (new object appears in the scene) we connect Source S to all detection nodes (these are the second node of each node split when we split the node of each object into two):

- Add connections that allow start of trajectory at "every" detection



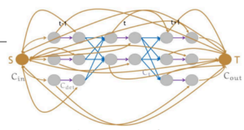
Similarly any object can dissapear at any time so we must connect the first node of an object to the output:

- Add connections that allow ending of a trajectory at "every" detection



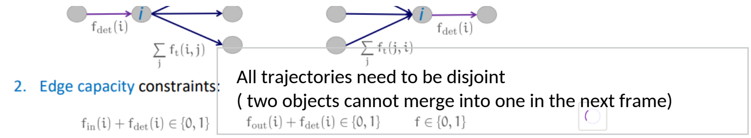
The constraints are:

Network flow formulation



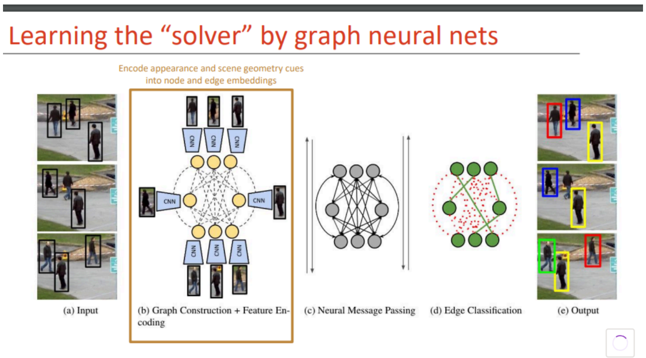
- Objective: $\mathcal{T}* = \arg\min_{\mathcal{T}} \sum_{i,j} C(i,j)f(i,j)$
- Constraints:
  1. Flow conservation at all nodes (all flow that goes into a node, goes out)

$$f_{in}(i) + f_{det}(i) = \sum_j f_t(i,j) \qquad \sum_j f_t(j,i) = f_{out}(i) + f_{det}(i)$$



We can write these constraints algebraically and get an NP-hard problem! To solve we relax the constraints and can rewrite the problem as a Linear Program.

In recent times, the "solver" can be learned by graph neural networks insetead:

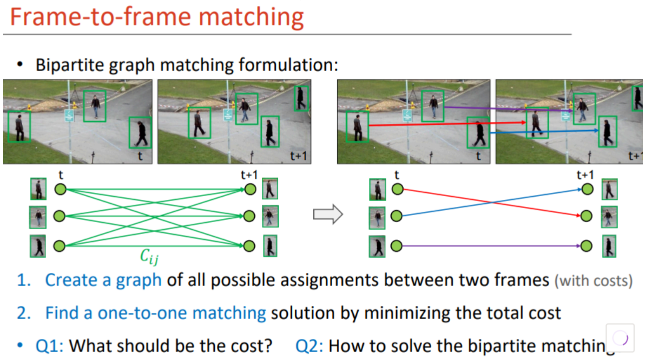


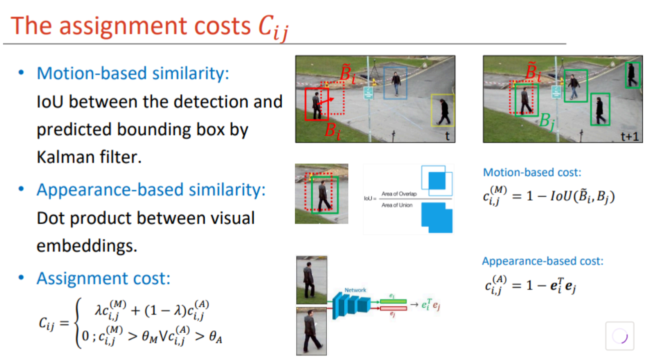
Online tracking:

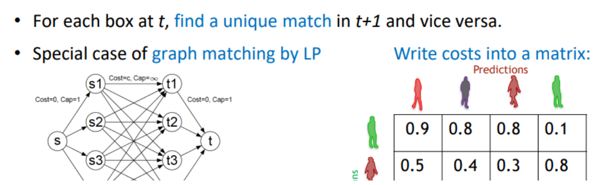
A Tracking iteration steps:

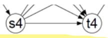
1. Run a detector on each new frame t+1
2. Match detections from frame t with the detections at frame t+1
3. Initialize/kill tracks (i.e., new objects entering, existing leaving)
4. Repeat from step (1) for all consecutive frames

Frame-to-frame matching









- Efficient algorithm exists:
  [1]Hungarian algorithm
  $x_{opt} = \underset{x}{\arg\min}\, c^T x$

Detectio

0.2	0.1	0.4	0.8
0.1	0.2	0.5	0.9

- What happens if we have missing prediction?
- What happens if there are many predictions, but some detections are not within the list of predictions?
- Introduce extra nodes that act as threshold on the acceptable assignment cost

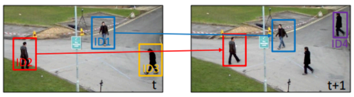
Write costs into a matrix:

Predictions

Detections

0.9	0.8	0.8	0.3	0.3
0.5	0.4	0.7	0.3	0.3
0.2	0.1	0.4	0.3	0.3
0.1	0.2	0.5	0.3	0.3
0.3	0.3	0.3	0.3	0.3

Track management (initalization / termination)



Classical example: DeepSORT

Wojke et al., Simple Online and Realtime Tracking with a Deep Association Metric, ICIP 2017

Managing unmatched predictions (example ID3):

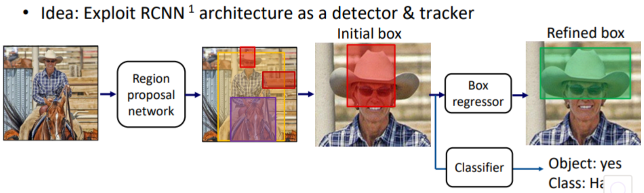
- Continue predicting by Kalman filter in the next frames.
- If not matched within the next T_{max} frames, terminate the trajectory.

Managing unmatched detections (example ID4):

- If detection score high, initialize a new track (new ID) and enter a probation period.
- If successfully matched for T_{min} consecutive frames, accept as a track.

Tracktor(++) [2]

- Recall R-CNN pipeline for object detection:
  - Propose regions
  - Refine and classify each region
- Idea: Exploit RCNN [1] architecture as a detector & tracker



Ren, He, Girshick, Sun. Faster R-CNN: Towards Real-Time Object Detection. NIPS 2015.

Tracktor(++)

- Predict each bounding box from *t* by a Kalman filter
- On image *t+1*, improve each translated box by the RCNN refinement head
- Run an RCNN detector to detect potential objects
- Associate the predicted boxes with detections & create new tracks

(Image stabilisation used to compensate for large between-frame camera motions.)