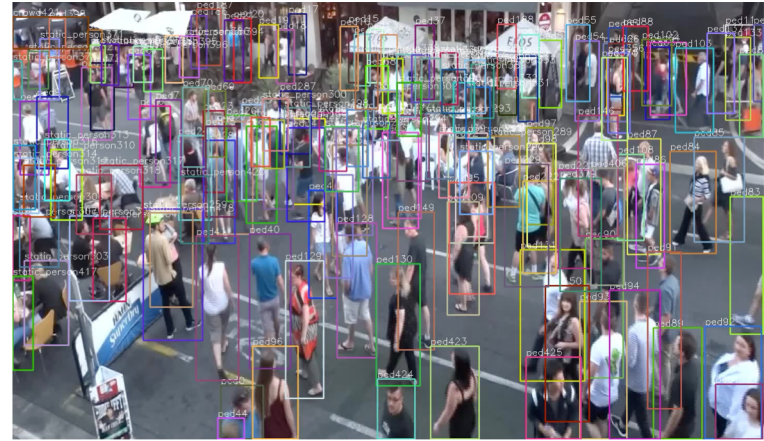


Multiple object tracking

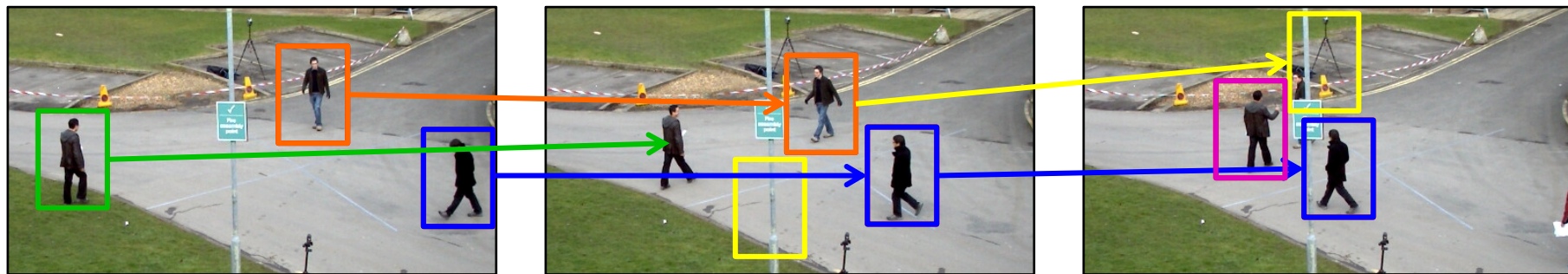
Different challenges

- Multiple objects of the same type
- Heavy occlusions
- Appearance is often very similar



Tracking-by-detection

- We will focus on algorithms where a set of detections is provided
 - Remember detections are not perfect!

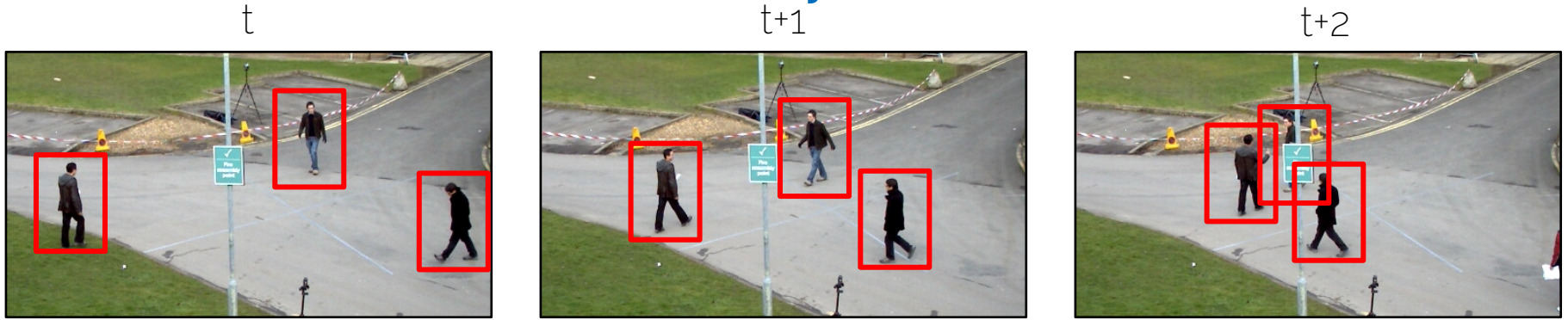


Find detections that match and form a trajectory

Online vs offline tracking

- Online tracking
 - Processes two frames at a time
 - For real-time applications
 - Prone to drifting → hard to recover from errors or occlusions
- Offline tracking
 - Processes a batch of frames
 - Good to recover from occlusions (short ones as we will see)
 - Not suitable for real-time applications
 - Suitable for video analysis

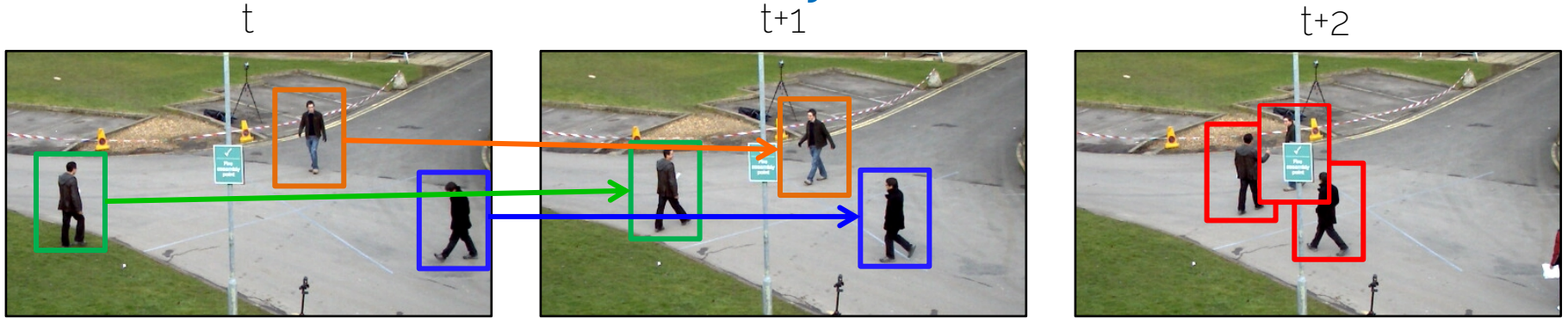
Frame-by-frame



- 1. Track initialization (e.g. using a detector)



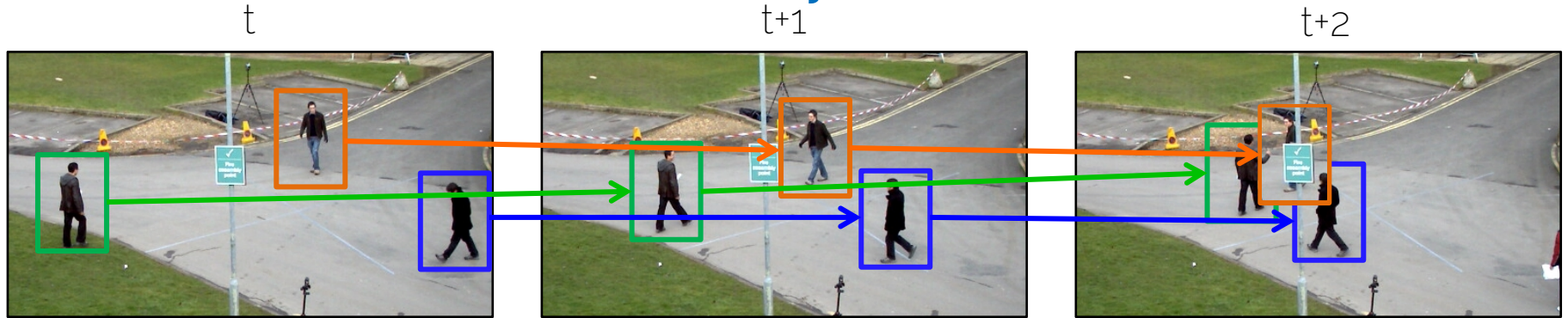
Frame-by-frame

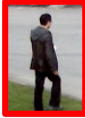


- 1. Track initialization (e.g. using a detector)
- 2. **Matching** detections at t with detections at $t+1$



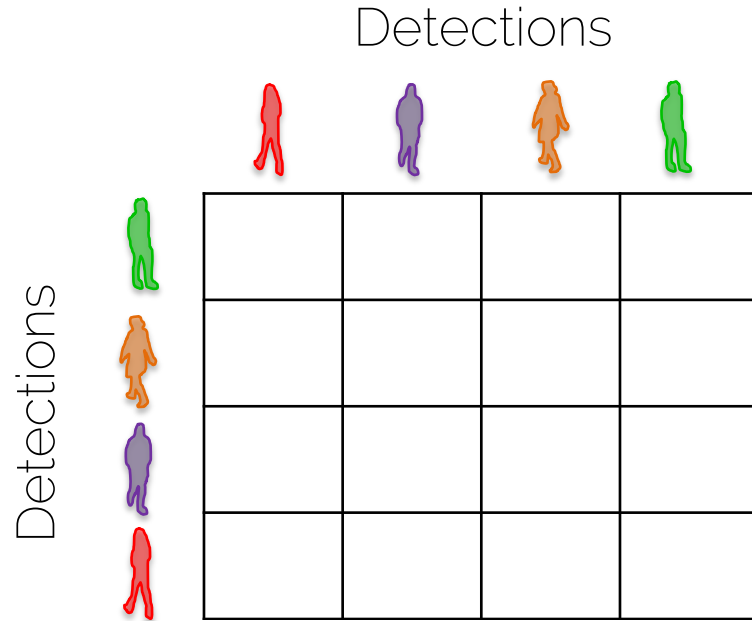
Frame-by-frame



- 1. Track initialization (e.g. using a detector) 
- 2. **Matching** detections at t with detections at $t+1$
- Repeat for every pair of frames

Frame-by-frame









- 3. Matching detections at t with detections at $t+1$



Frame-by-frame

- Bipartite matching
 - Define distances between boxes
(e.g., IoU, pixel distance, 3D distance)

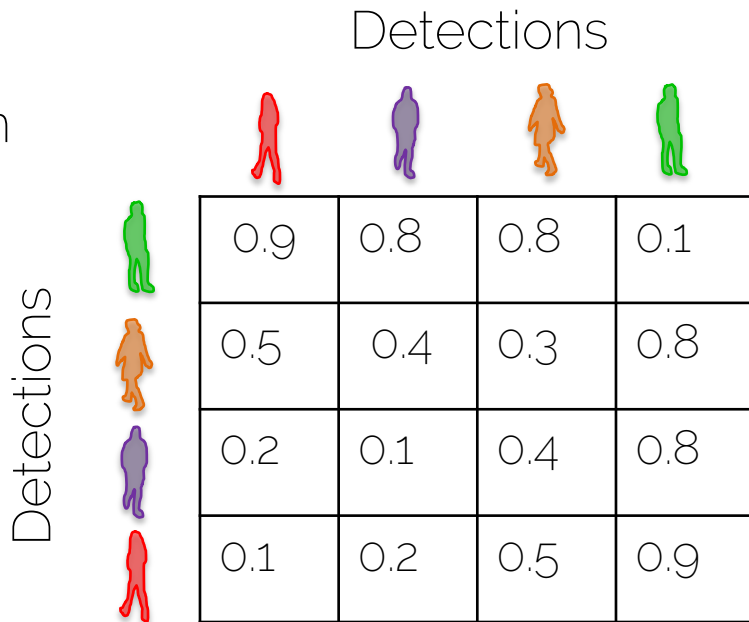
Detections









				
 Detections	0.9	0.8	0.8	0.1
 Detections	0.5	0.4	0.3	0.8
 Detections	0.2	0.1	0.4	0.8
 Detections	0.1	0.2	0.5	0.9

Frame-by-frame

- Bipartite matching
 - Define distances between boxes
(e.g., IoU, pixel distance, 3D distance)
 - Solve the unique matching with e.g., the Hungarian algorithm*

Detections











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

*Demo: <http://www.hungarianalgorithm.com/solve.php>

Frame-by-frame

- Bipartite matching
 - Define distances between boxes
(e.g., IoU, pixel distance, 3D distance)
 - Solve the unique matching with e.g., the Hungarian algorithm*
 - Solutions are the unique assignments that minimize the total cost

Detections

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

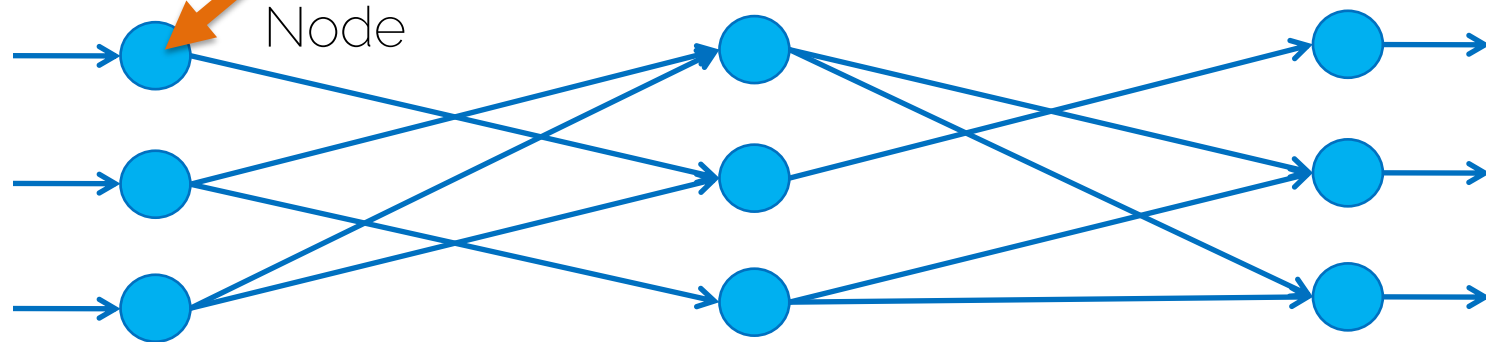
*Demo: <http://www.hungarianalgorithm.com/solve.php>

Frame-by-frame

- Problems with frame-by-frame tracking
 - Cannot recover from errors. If a detection is missing from a frame, we have to end the trajectory.
 - All decisions are essentially local
 - Hard to recover from errors in the detection step
- Solution: find the minimum cost solution for ALL frames and ALL trajectories

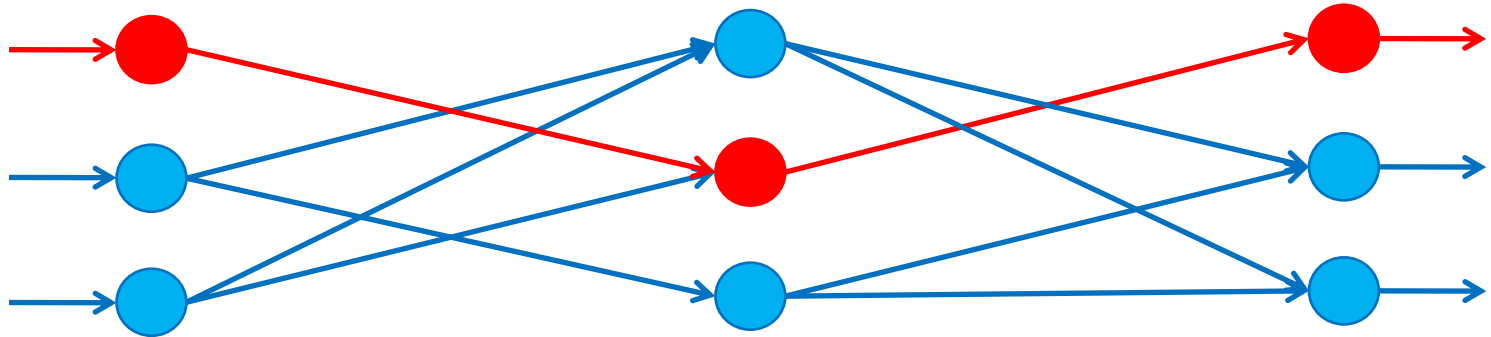
Graph-based MOT

Tracking with network flows



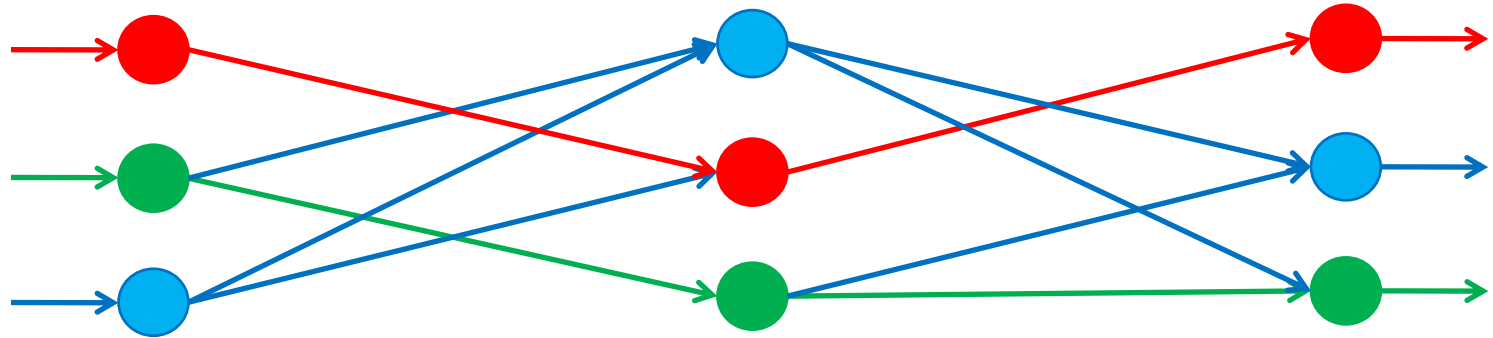
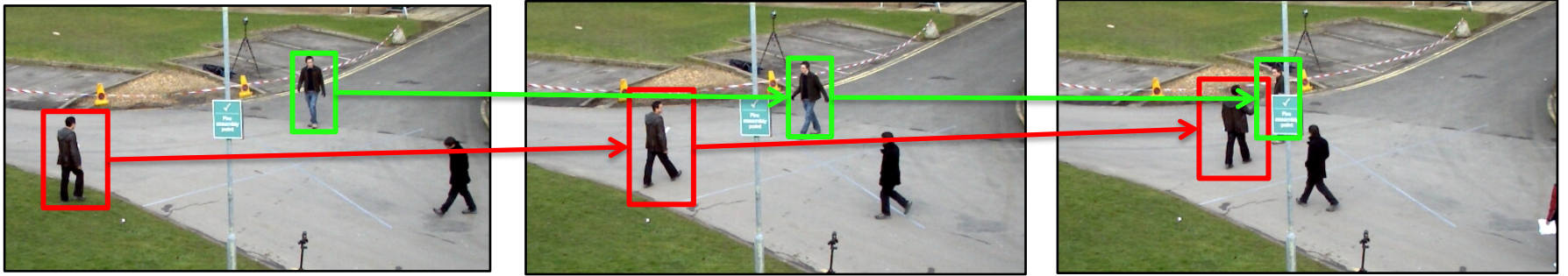
Graphical model

Tracking with network flows



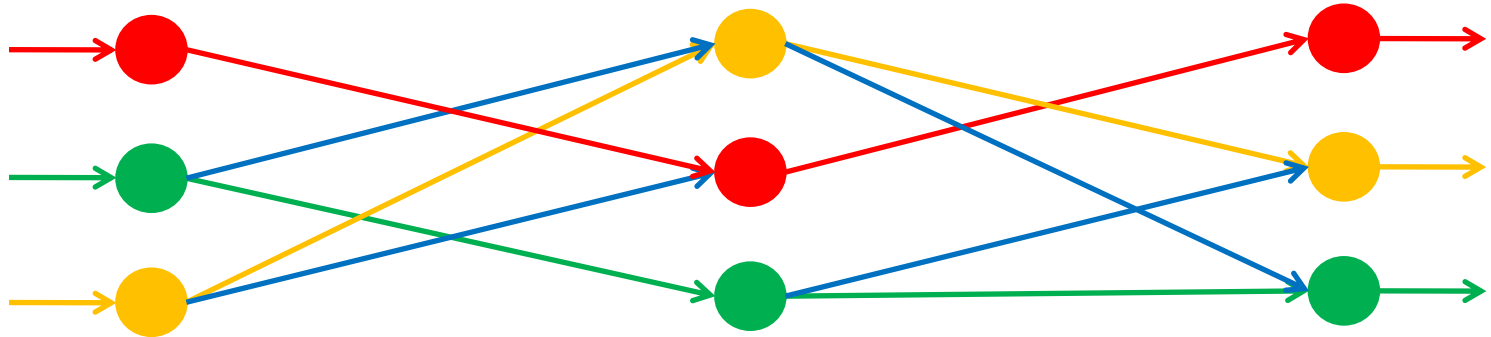
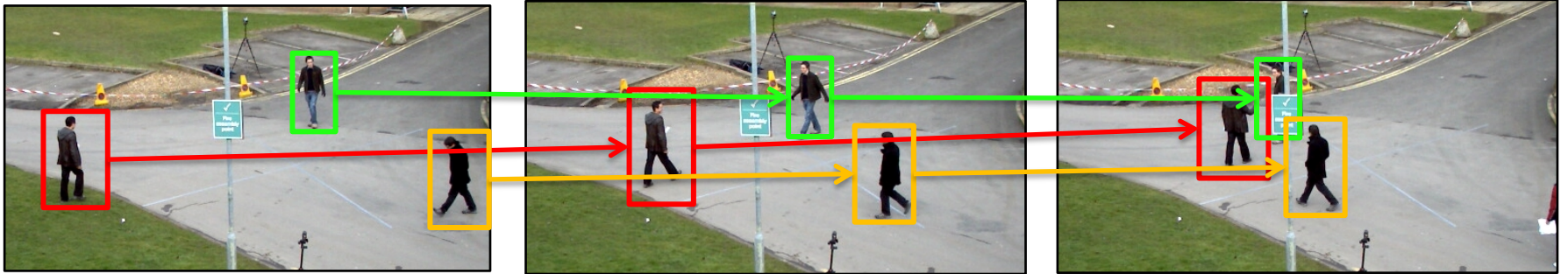
L. Leal-Taixé et al. "Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker." ICCVW2011

Tracking with network flows



L. Leal-Taixé et al. "Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker." ICCVW2011

Tracking with network flows



L. Leal-Taixé et al. "Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker." ICCVW2011

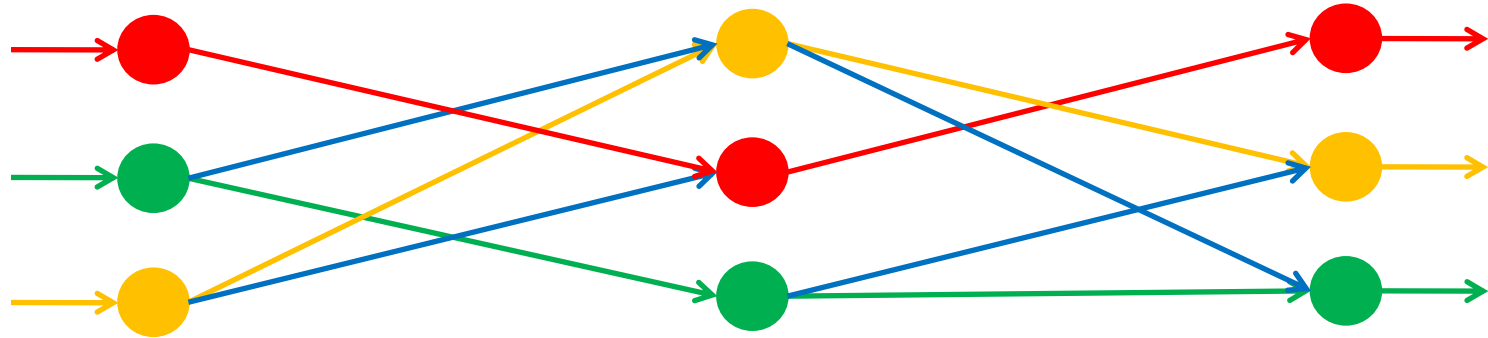
Tracking with network flows

- Node = detection
- Edge = flow = trajectory
- 1 unit of flow = 1 pedestrian

Tracking with network flows

- Solving the Minimum Cost Flow Problem

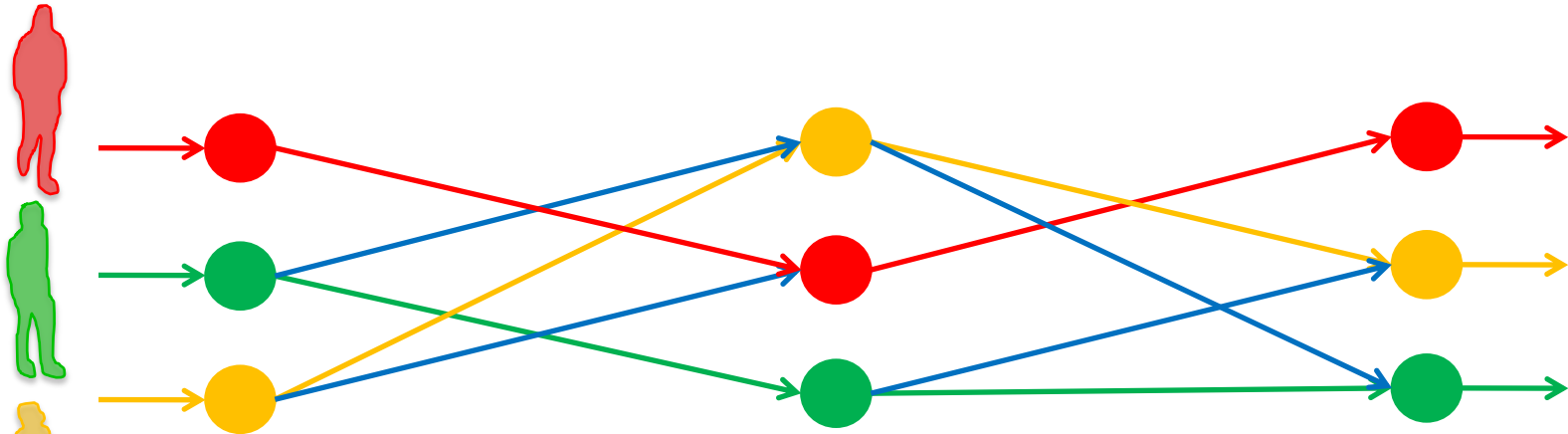
"Determine the minimum cost of shipment of a commodity through a network"



Tracking with network flows

- Solving the Minimum Cost Flow Problem

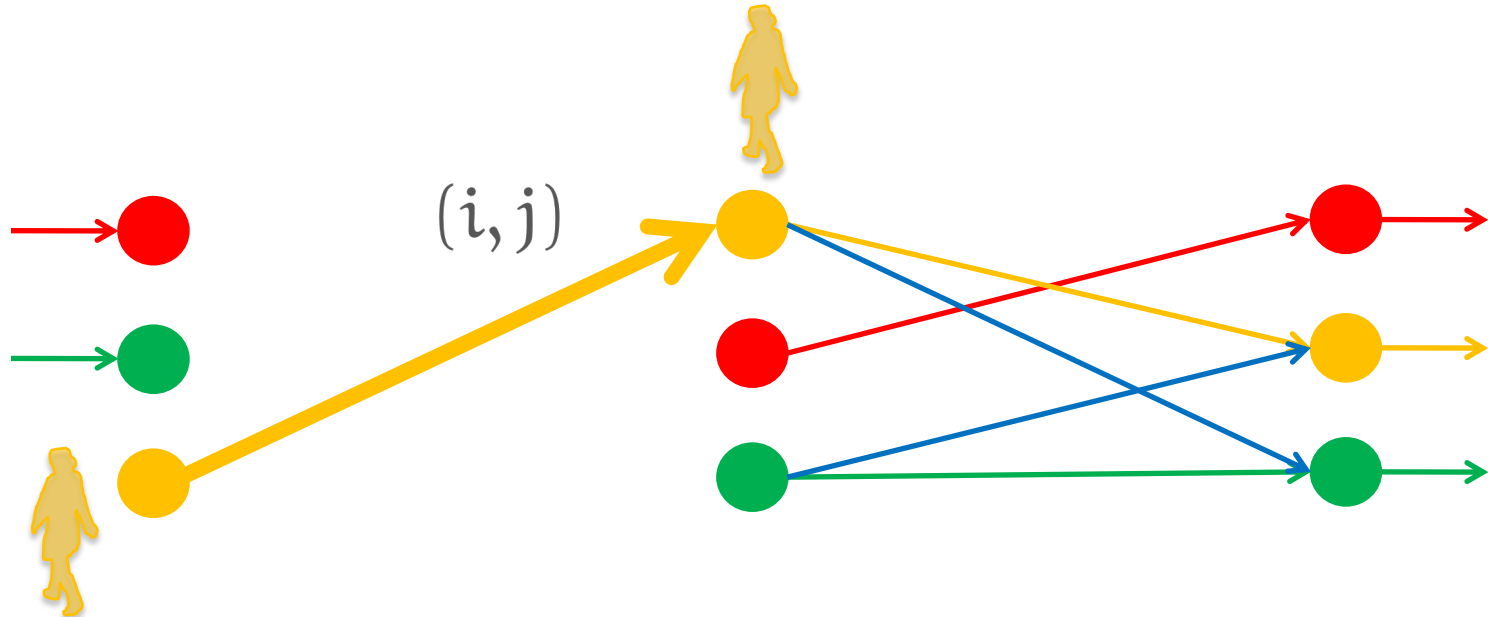
"Determine the minimum cost of shipment of a commodity through a network"



Ravindra K. Ahuja; Thomas L. Magnanti & James B. Orlin. "Network Flows: Theory, Algorithms, and Applications". Prentice-Hall, Inc. 1993

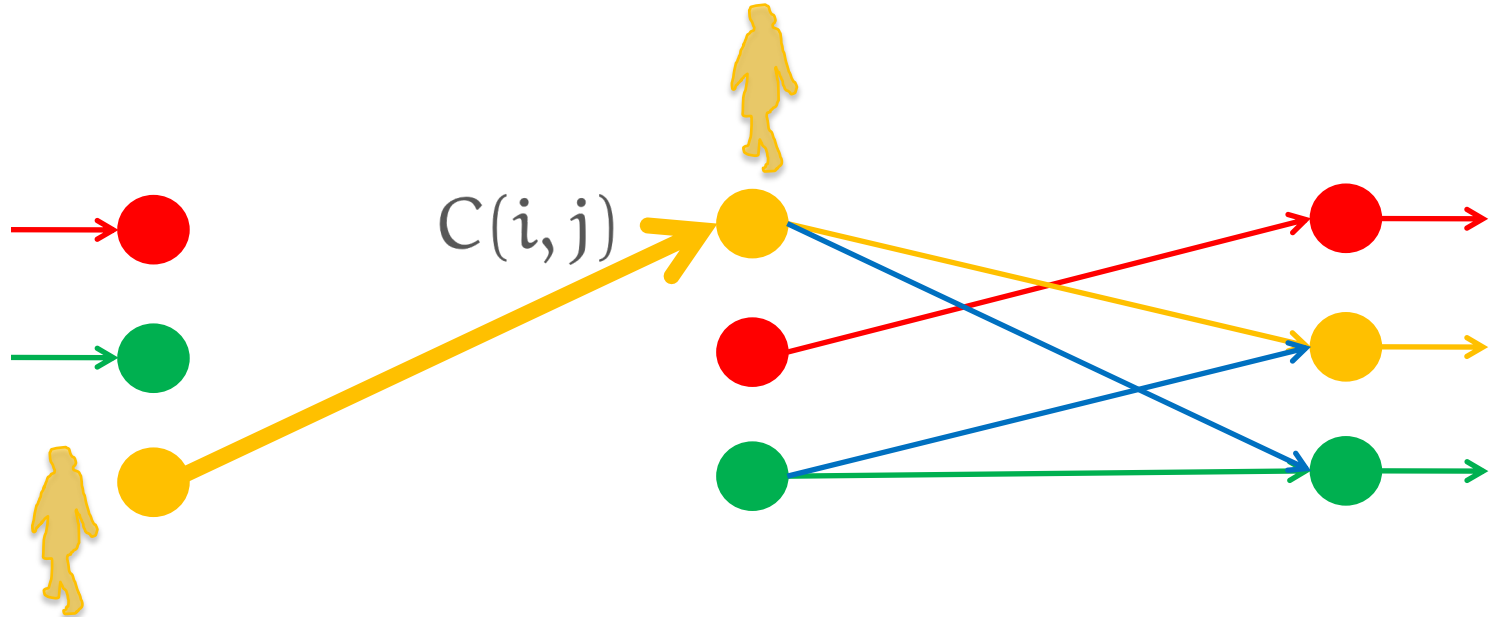
Tracking with network flows

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



Tracking with network flows

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

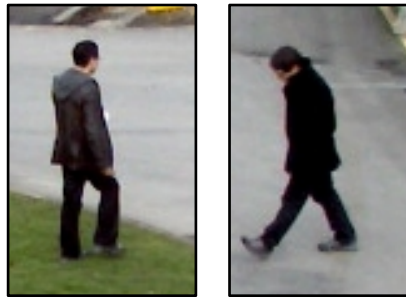


Tracking with network flows

- Objective function



↓ C



↑ C

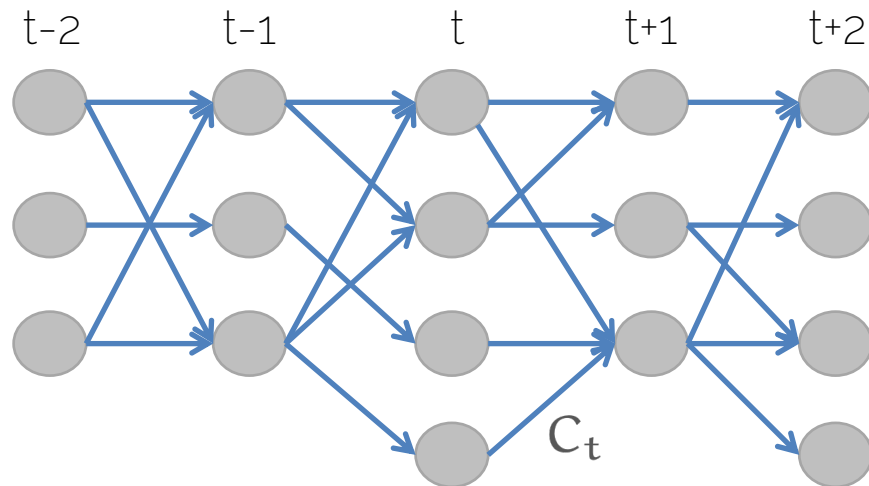
$$\mathcal{T}^* = \arg \min_{\mathcal{T}} \sum_{i,j} C(i,j) f(i,j)$$

Optimal set of trajectories

Indicator {0,1}

Costs – what will drive the tracking

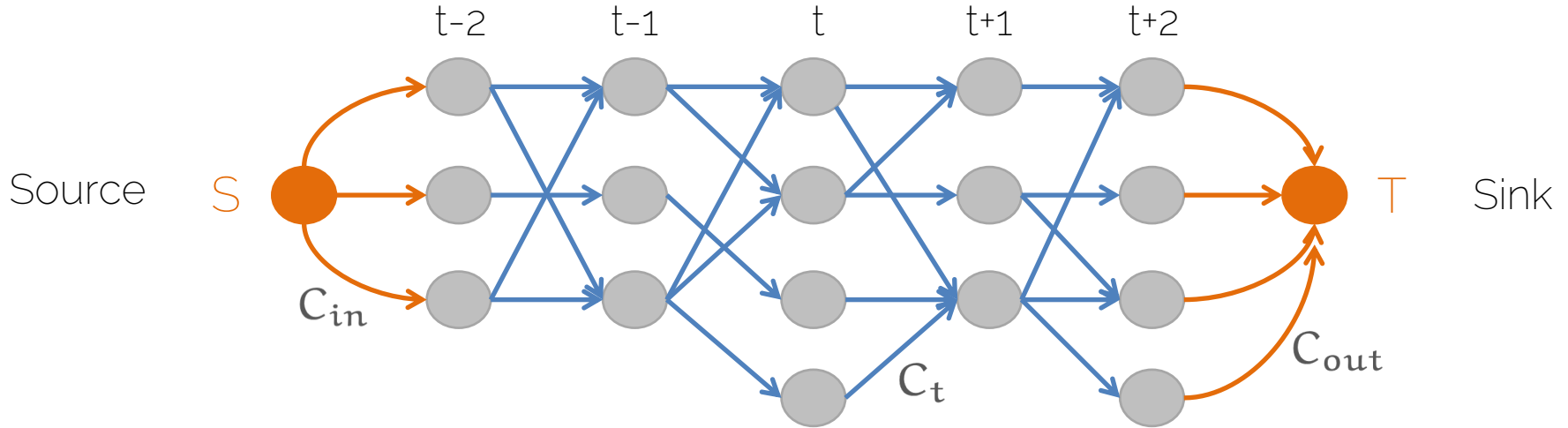
Tracking with network flows



Transition: cost \propto distance between detections

FLOW = TRAJECTORY = PEDESTRIAN

Tracking with network flows

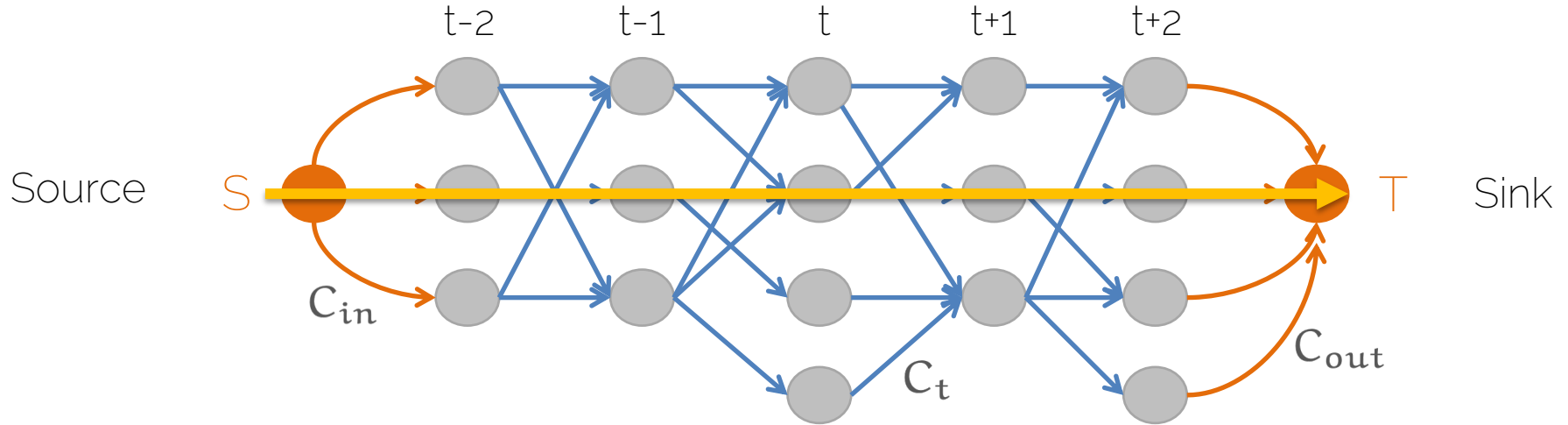


Entrance/exit: cost to start or end a trajectory

Transition: cost \propto distance between detections

FLOW = TRAJECTORY = PEDESTRIAN

Tracking with network flows

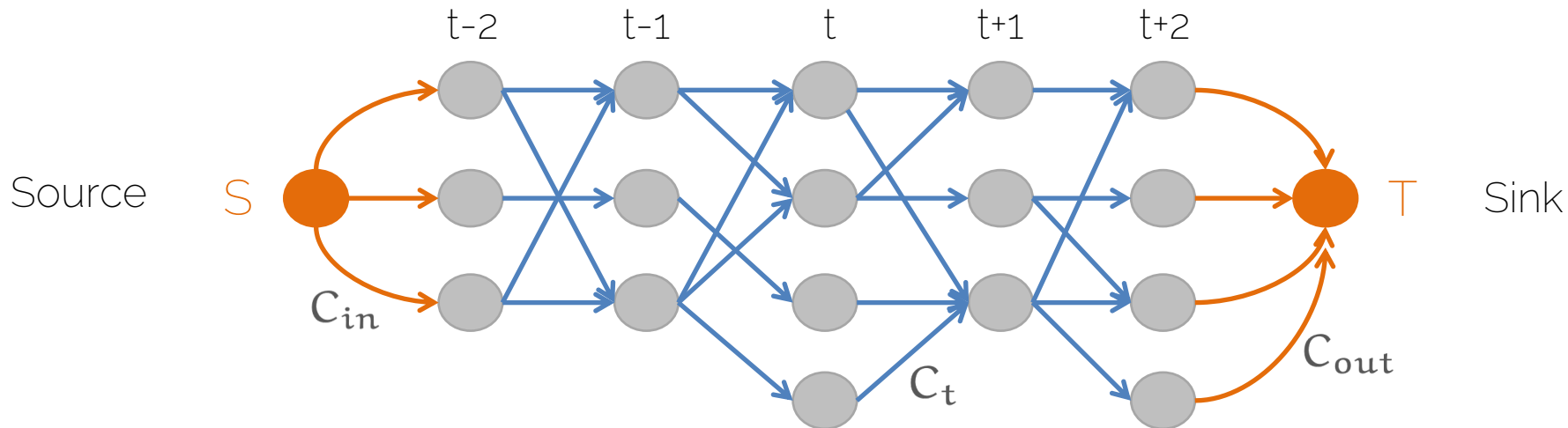


Entrance/exit: cost to start or end a trajectory

Transition: cost \propto distance between detections

Flow can only start at the source node and end at the sink node

Tracking with network flows

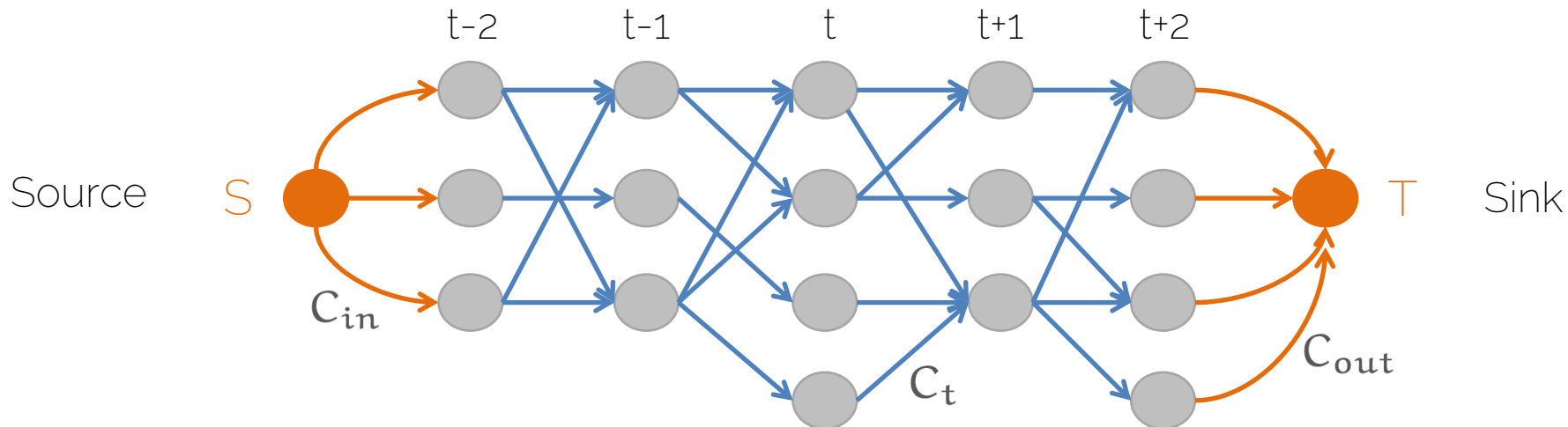


Entrance/exit: cost to start or end a trajectory

Transition: cost \propto distance between detections

What happens if all costs are positive? $\mathcal{T}^* = \arg \min_{\mathcal{T}} \sum_{i,j} C(i,j) f(i,j)$

Tracking with network flows



Entrance/exit: cost to start or end a trajectory

Transition: cost \propto distance between detections

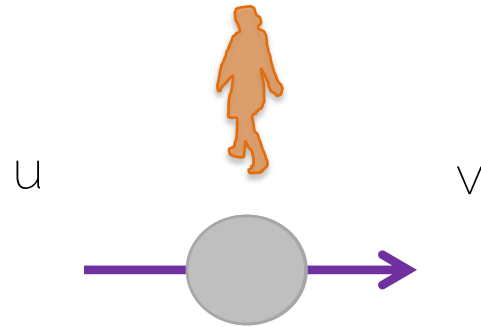
What happens if all costs are positive? $\mathcal{T}^* = \arg \min_{\mathcal{T}} \sum_{i,j} C(i,j) f(i,j)$

Trivial solution: zero flow!

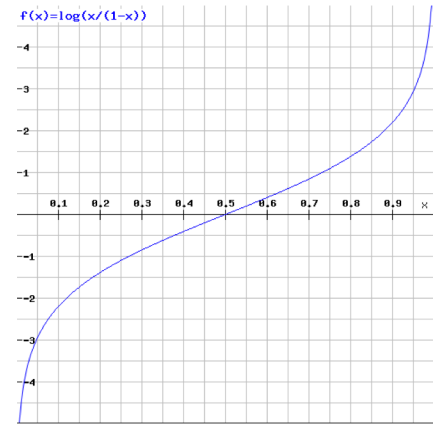
Tracking with network flows

- We need a negative cost

Detection edge

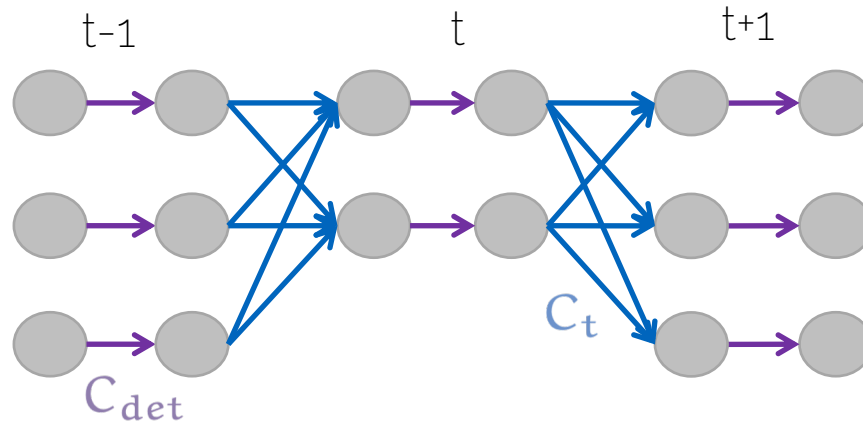


$$C_{\text{det}} = \log \frac{\beta_i}{1 - \beta_i}$$



Probability that detection i is a false alarm

Complete graph



Complete graph

Source

S

C_{in}

$t-1$

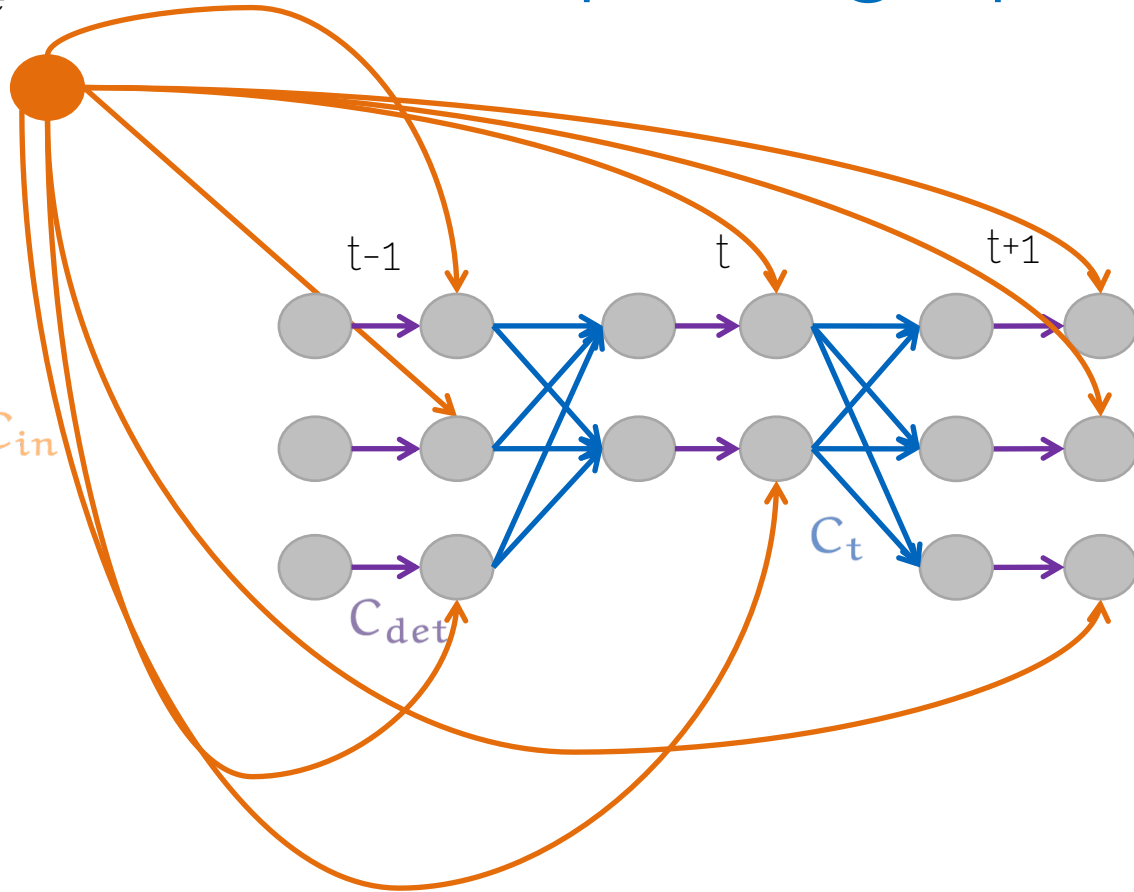
t

$t+1$

C_{det}

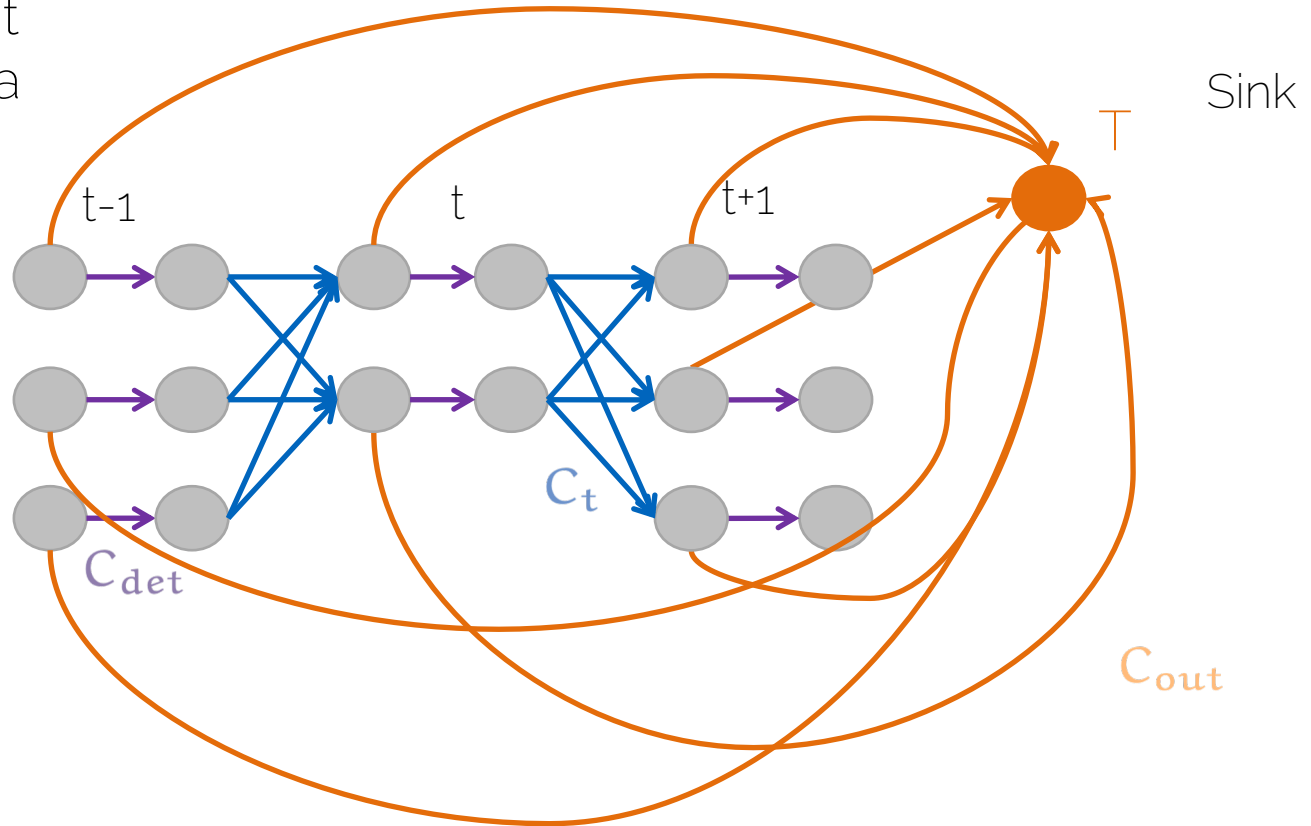
C_t

Connections that allow us to start a trajectory



Complete graph

Connections that allow us to end a trajectory



Linear Program MOT formulation

Why a linear program?

- Fast solvers (e.g., Simplex algorithm)
- Guaranteed to converge to the global optimum

Minimum cost flow problem

- Objective function

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

Constraints

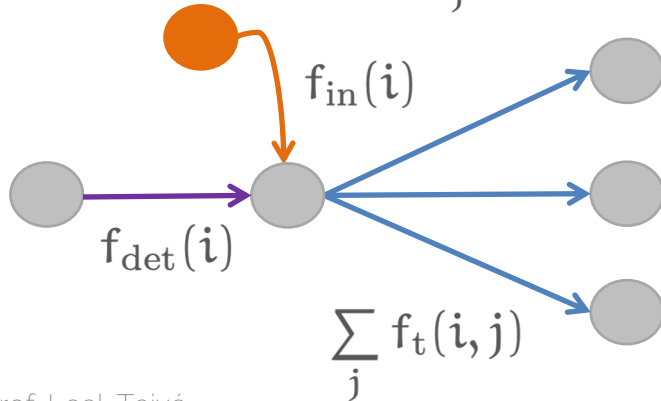
- Objective function

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

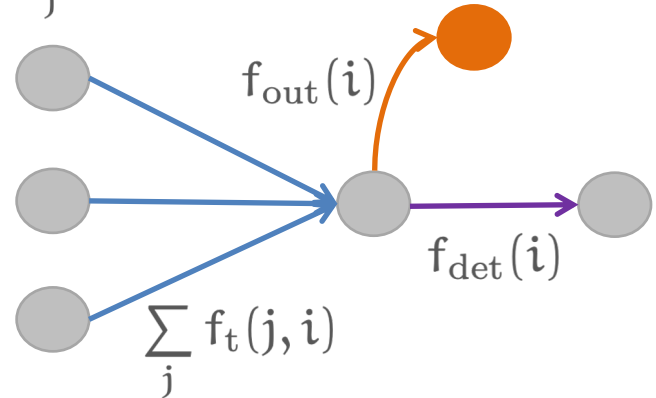
- Subject to

Flow conservation at the nodes

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



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



Constraints

- Objective function

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

- Subject to

Flow conservation at the nodes

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

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

Edge capacities

NP-hard!!

$$f_{\text{in}}(\mathbf{i}) + f_{\text{det}}(\mathbf{i}) \in \{0, 1\}$$

$$f_{\text{out}}(\mathbf{i}) + f_{\text{det}}(\mathbf{i}) \in \{0, 1\}$$

$$f \in \{0, 1\}$$

LP relaxation

- Objective function

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

- Subject to

Flow conservation at the nodes

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

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

Edge capacities

LP-relaxation

$$0 \leq f_{\text{in}}(\mathbf{i}) + f_{\text{det}}(\mathbf{i}) \leq 1$$

$$0 \leq f_{\text{out}}(\mathbf{i}) + f_{\text{det}}(\mathbf{i}) \leq 1$$

$$0 \leq f \leq 1$$

Solver

- Objective function

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

Given the shape of the constraints (total unimodularity), we solve the relaxed problem and still get **integer** solutions.

Objective function

- Objective function

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

Objective function

- Objective function

$$\mathcal{T}^* = \arg \min_{\mathcal{T}} \sum_i C_{\text{in}}(\mathbf{i}) f_{\text{in}}(\mathbf{i}) + \sum_{\mathbf{i}, \mathbf{j}} C_t(\mathbf{i}, \mathbf{j}) f_t(\mathbf{i}, \mathbf{j}) \\ + \sum_i C_{\text{det}}(\mathbf{i}) f_{\text{det}}(\mathbf{i}) + \sum_i C_{\text{out}}(\mathbf{i}) f_{\text{out}}(\mathbf{i})$$



$$C(\mathbf{i}) = -\log P(\mathbf{i})$$

- Equivalent to Maximum a-posteriori tracking formulation

$$\mathcal{T}^* = \arg \max_{\mathcal{T}} \prod_j P(\mathbf{o}_j | \mathcal{T}) P(\mathcal{T})$$

Two ways forward

- 1. Improving the costs (aka more learning)
 - L. Leal-Taixé et al. "Learning an image-based motion context for multiple people tracking". CVPR 2014.
 - L. Leal-Taixé et al. "Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker". ICCVW 2011
 - L. Leal-Taixé et al. "Learning by tracking: Siamese CNN for robust target association". CVPRW 2016.
 - S. Schulter et al. „Deep network flow for multi-object tracking". CVPR 2017.
 - J. Son et al. „Multi-object tracking with quadruplet convolutional neural networks". CVPR 2017.
 - Ristani and Tomasi. "Features for multi-target multi-camera tracking and re-identification". CVPR 2018
 - J. Xu et al. „Spatial-temporal relation networks for multi-object tracking". ICCV 2019.

Two ways forward

- 2. Making the graph more complex (including more connections)
 - M. Keuper et al. „Motion segmentation and multiple object tracking by correlation co-clustering“. PAMI 2018.
 - S. Tang et al. „Subgraph decomposition for multi-target tracking:“. CVPR 2015.
 - S. Tang et al. „Multiple people tracking by lifted multicut and person reidentification“. CVPR 2017

More learning

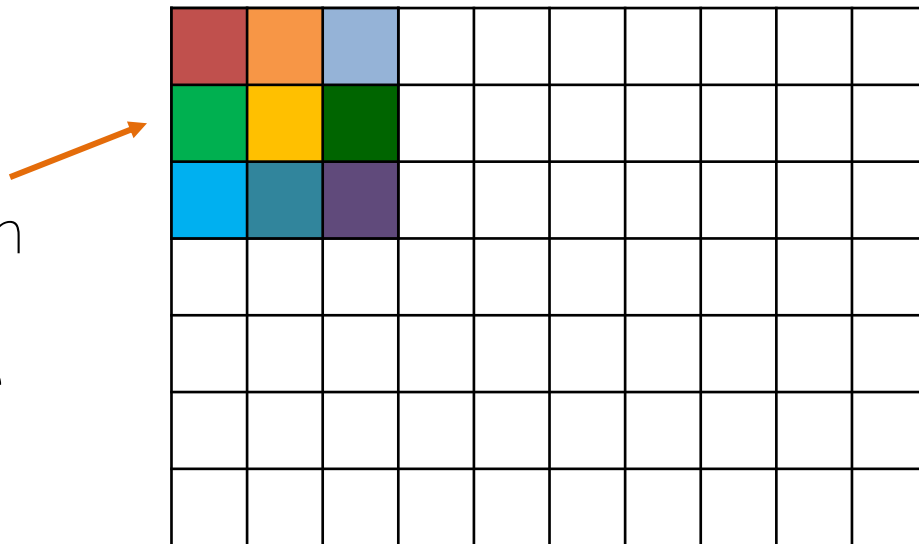
- Can we learn:
 - Features for multi-object tracking (e.g., costs)
 - To do data association, i.e., find a solution on the graph
- We will exploit the graph structure we have just seen and perform learning directly on the graph

Deep Learning on graphs

The domain so far

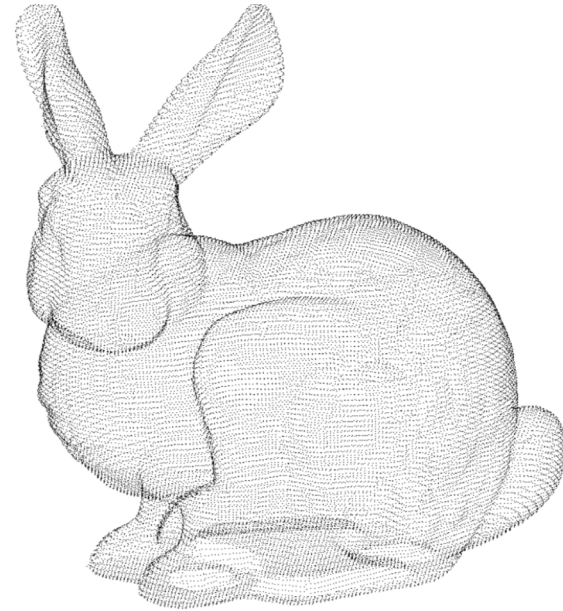
- Regularity on the domain
 - Order of the pixels is important

Your convolution filter imposes a certain structure



A new domain

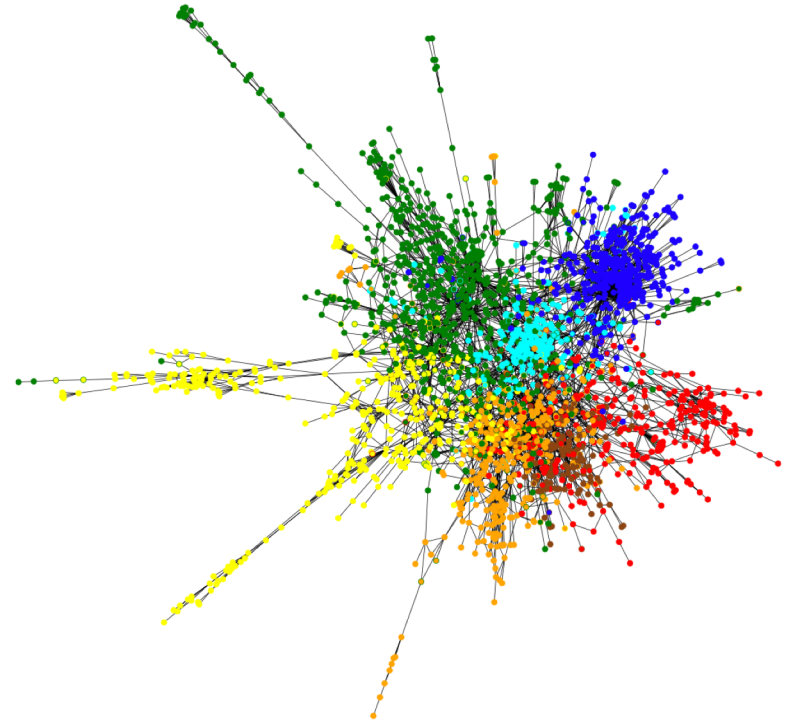
- Rich Information
 - 3D Point Location
 - Other features:
 - RGB/ Intensity
 - Semantic
- Irregularity
 - Permutation Invariance
 - Transformation Invariance



Point Cloud

A new domain

- A citation network
 - Each node is a paper
 - Connection is a citation
- Similar for:
 - Social networks
 - Recommender systems

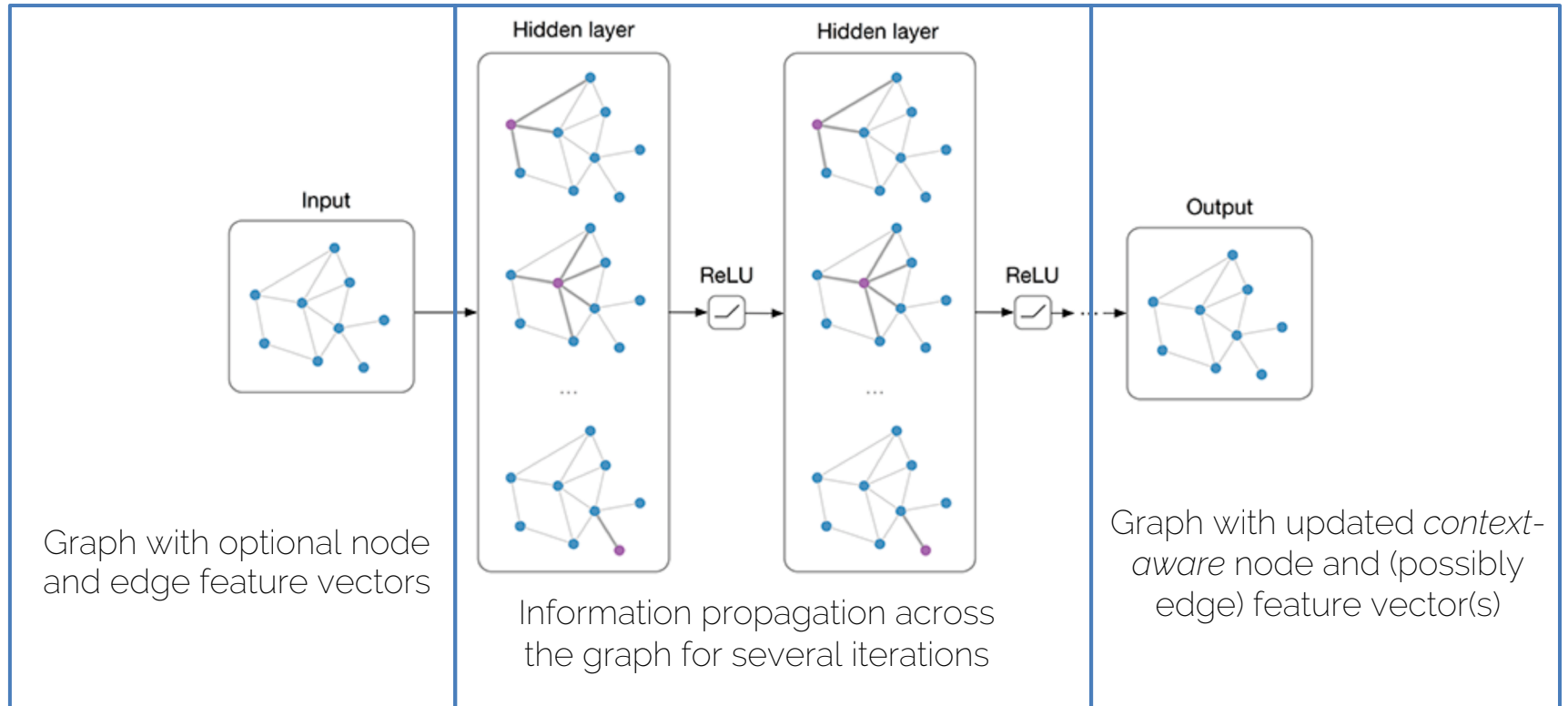


M. Bronstein et al. „Geometric deep learning: going beyond Euclidean data“. IEEE Signal Processing Magazine. 2017

Deep learning on graphs

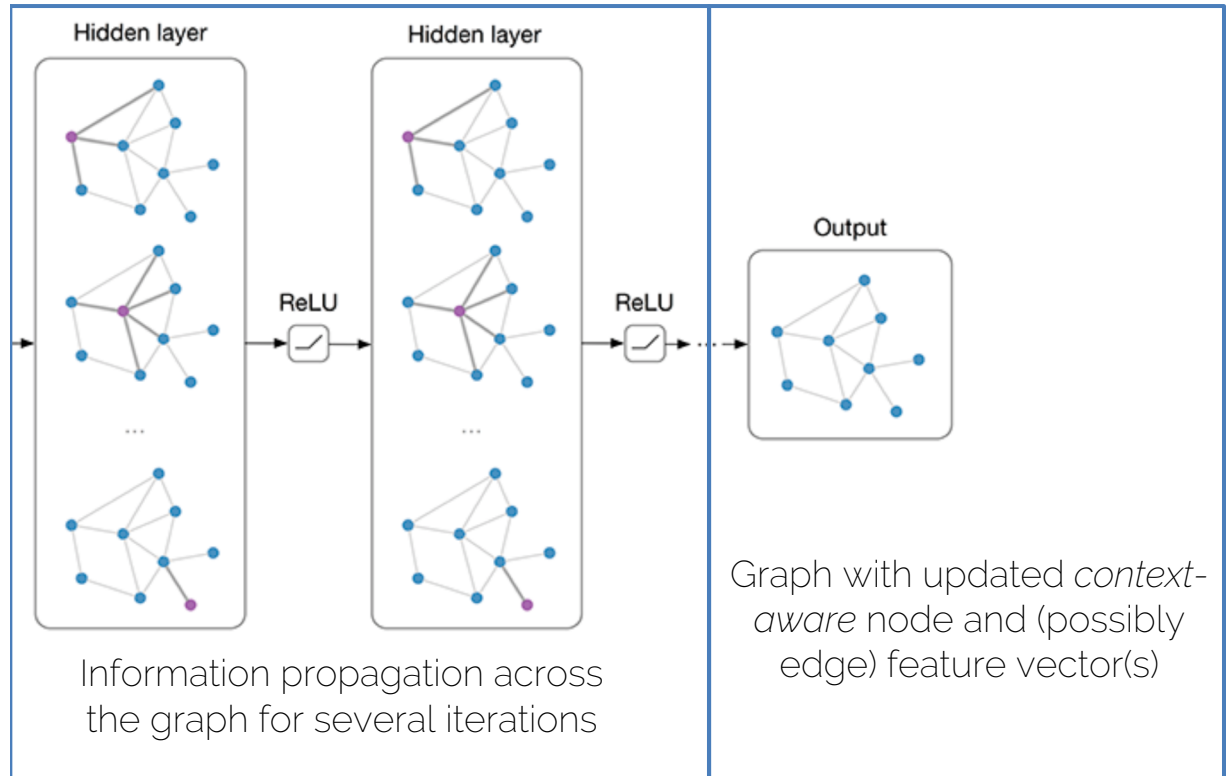
- Generalizations of neural networks that can operate on graph-structured domains:
 - Scarselli et al. "The Graph Neural Network Model". IEEE Trans. Neur. Net 2009.
 - Kipf et al. "Semi-Supervised Classification with Graph Convolutional Networks. ICLR 2016.
 - Gilmer et al. "Neural Message Passing for Quantum Chemistry". ICML 2017
 - Battaglia et al. "Relational inductive biases, deep learning, and graph networks". arXiv 2018 (review paper)
- Key challenges:
 - Variable sized inputs (number of nodes and edges)
 - Need **invariance to node permutations**

General Idea

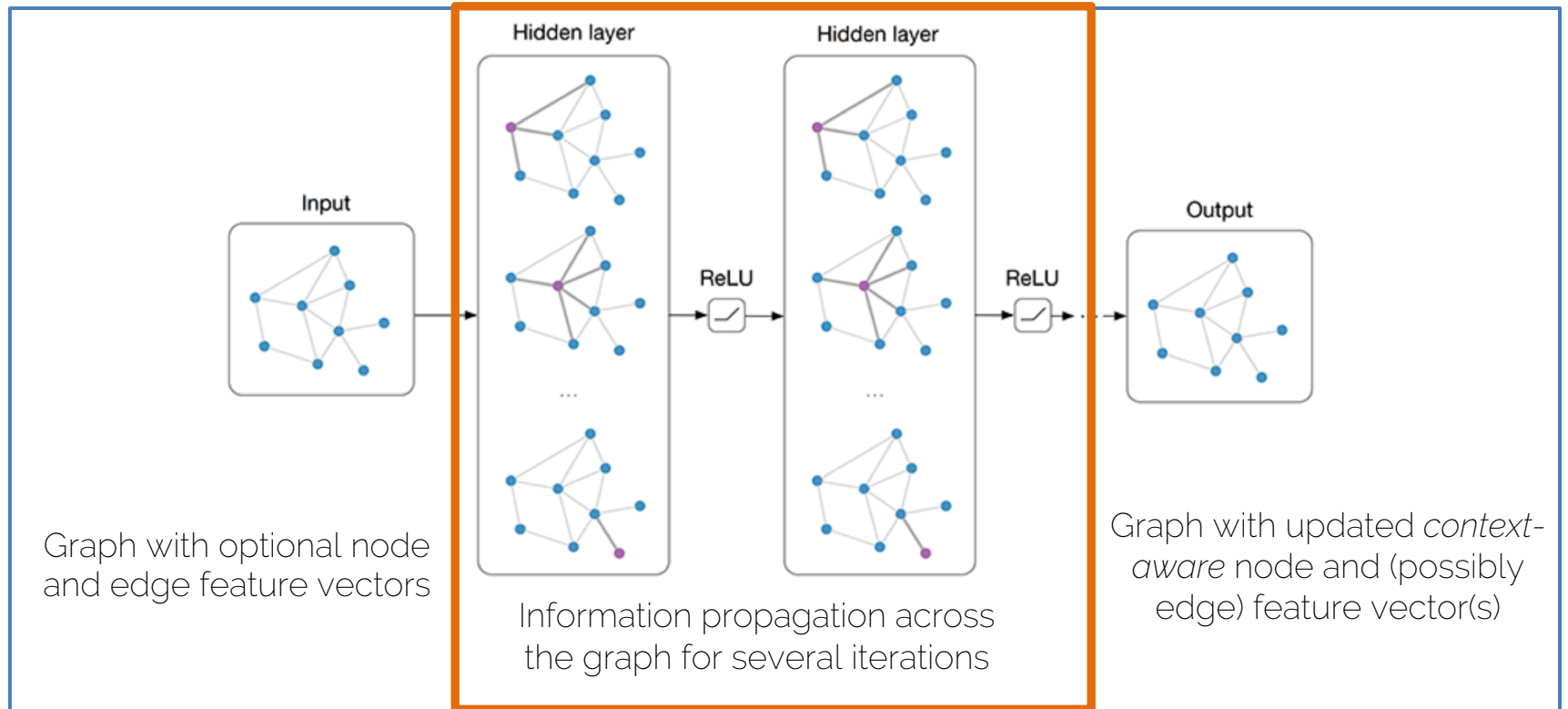


General Idea

Each update step is understood as a “layer” in common NNs

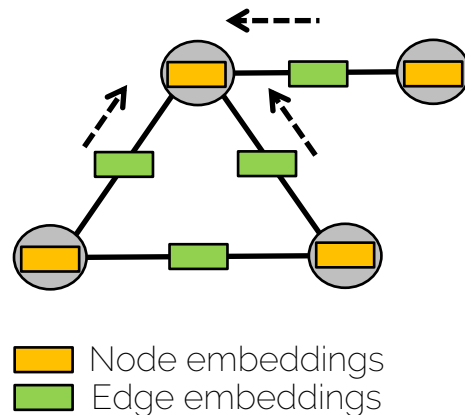


General Idea



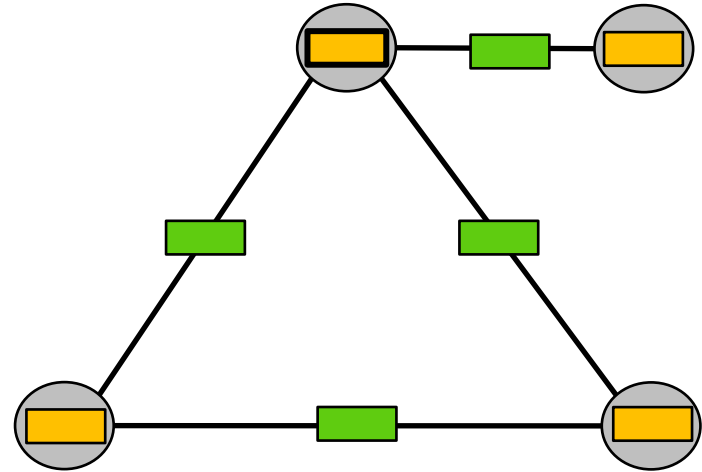
Neural Message Passing

- Notation:
 - Graph: $G = (V, E)$
 - Initial embeddings: $h_{(i,j)}^{(0)}, (i,j) \in E$ $h_i^{(0)}, i \in V$
 - Node embeddings after l steps: $h_i^{(l)}, i \in V$
- Goal:
 - Encode contextual graph information in node embeddings by iteratively combining neighboring nodes' features



Neural Message Passing

- At every iteration, every node receives features from its neighboring nodes.
- These features are then aggregated with an order invariant operation and combined with the current features with a learnable function



Neural Message Passing

- At every message passing step l , for every node do:

$$m_v^{(l+1)} = \sum_{u \in N_v} \underbrace{M^{(l)}}_{\text{Learnable function (e.g. MLP) with shared weights across the entire graph}}(h_u^{(l)}, h_v^{(l)}, h_{(u,v)}^{(0)})$$

Message

Aggregation overall all neighbors

Neural Message Passing

- At every message passing step l , for every node do:

$$m_v^{(l+1)} = \sum_{u \in N_v} M^{(l)}(h_u^{(l)}, h_v^{(l)}, h_{(u,v)}^{(0)})$$

$$h_v^{(l+1)} = \underbrace{U^{(l)}}_{\text{Learnable function (e.g. MLP) with shared weights across the entire graph}}(h_v^{(l)}, m_v^{(l+1)})$$

Embedding
update

Learnable function (e.g. MLP) with
shared weights across the entire graph

Neural Message Passing

- At every message passing step l , for every node do:

$$m_v^{(l+1)} = \sum_{u \in N_v} M^{(l)}(h_u^{(l)}, h_v^{(l)}, h_{(u,v)}^{(0)})$$

$$h_v^{(l+1)} = U^{(l)}(h_v^{(l)}, m_v^{(l+1)})$$

Most Graph Neural Network Models can be seen as specific example of this formulation

Neural Message Passing: An Example

$$m_v^{(l+1)} = \underbrace{\sum_{u \in N_v} \frac{h_u^{(l)}}{|N_v|}}_{\text{Average neighbors' feature embeddings}}$$

Average neighbors' feature embeddings

Neural Message Passing: An Example

$$h_v^{(l+1)} = \sigma \left(W^{(l+1)} m_v^{(l+1)} + B^{(l+1)} h_v^{(l)} \right)$$

New message

Previous embedding

Non-linearity

Learnable matrices, shared for all nodes

Combine node features with its neighbors'

The diagram shows the equation for the next layer's node embedding. The term $W^{(l+1)} m_v^{(l+1)}$ is labeled as the 'New message'. The term $B^{(l+1)} h_v^{(l)}$ is labeled as 'Combine node features with its neighbors'. The activation function σ is labeled as 'Non-linearity'. The matrices $W^{(l+1)}$ and $B^{(l+1)}$ are collectively labeled as 'Learnable matrices, shared for all nodes'. The term $h_v^{(l)}$ is labeled as the 'Previous embedding'.

Neural Message Passing: An Example

- We can use MLPs or even recurrent networks, instead of linear functions
- These are THE SAME for ALL nodes and edges!

$$h_v^{(l+1)} = \sigma \left(\underbrace{MLP_1^{(l+1)}}_{\text{MLP}_1^{(l+1)}} m_v^{(l+1)} + \underbrace{MLP_2^{(l+1)}}_{\text{MLP}_2^{(l+1)}} h_v^{(l)} \right)$$

Graph Convolutional Networks


$$m_v^{(l+1)} = \sum_{u \in N_v \cup \{v\}} \frac{h_u^{(l)}}{\sqrt{|N_v| |N_u|}}$$

Self loop Per neighbor degree normalization

Kipf and Welling. "Semi-Supervised Classification with Graph Convolutional Networks". ICLR 2016.

Graph Convolutional Networks

$$m_v^{(l+1)} = \sum_{u \in N_v \cup \{v\}} \frac{h_u^{(l)}}{\sqrt{|N_v| |N_u|}}$$


$$h_v^{(l+1)} = \sigma \left(W^{(l+1)} m_v^{(l+1)} \right)$$


Same learnable matrix for self-loops and regular neighbors

Kipf and Welling. "Semi-Supervised Classification with Graph Convolutional Networks". ICLR 2016.

Graph Convolutional Networks

$$m_v^{(l+1)} = \sum_{u \in N_v \cup \{v\}} \frac{h_u^{(l)}}{\sqrt{|N_v| |N_u|}}$$


$$h_v^{(l+1)} = \sigma \left(W^{(l+1)} m_v^{(l+1)} \right)$$


Matrix of weights is of size = #channels out x #channels in

Kipf and Welling. "Semi-Supervised Classification with Graph Convolutional Networks". ICLR 2016.

Graph Convolutional Networks

- We want to collect information from our neighbors and convert it to a new embedding

$$h_v^{(l+1)} = \sigma \left(W^{(l+1)} m_v^{(l+1)} \right)$$


Matrix of weights is of size = #channels out x #channels in

Graph Convolutional Networks

$$m_v^{(l+1)} = \sum_{u \in N_v \cup \{v\}} \frac{h_u^{(l)}}{\sqrt{|N_v| |N_u|}}$$

- Unlike a normal image convolutional filter, here the neighbors are not regular (as they are in the image space), hence I have to do a permutation-invariant aggregation operation before the convolution.

Graph Convolutional Networks

$$m_v^{(l+1)} = \sum_{u \in N_v \cup \{v\}} \frac{h_u^{(l)}}{\sqrt{|N_v| |N_u|}}$$

Aggregation

$$h_v^{(l+1)} = \sigma \left(W^{(l+1)} m_v^{(l+1)} \right)$$

Convolution

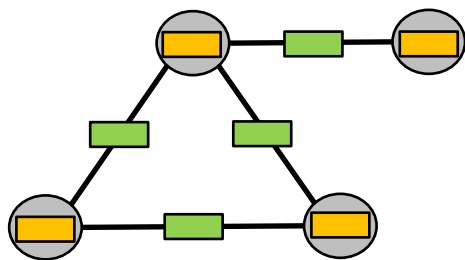
Kipf and Welling. "Semi-Supervised Classification with Graph Convolutional Networks". ICLR 2016.

What About Edge Embeddings?

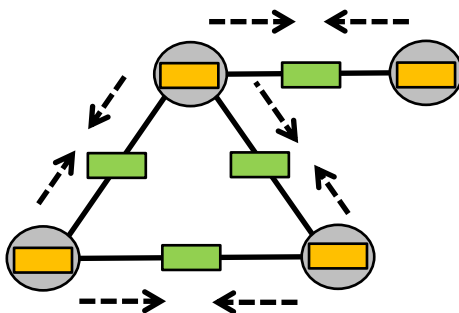
- The framework we've presented is only suited to learn node embeddings. But what happens if our focus is on edge features?
- At least, two options:
 - Work on the 'dual' or 'line' graph
 - E.g. Chen et al. "Supervised Community Detection with Line Graph Neural Networks", ICLR 2019.
 - Use a more general formulation that admits edge updates
 - E.g. Battaglia et al. "Relational inductive biases, deep learning, and graph networks". arXiv 2018

A More General Framework

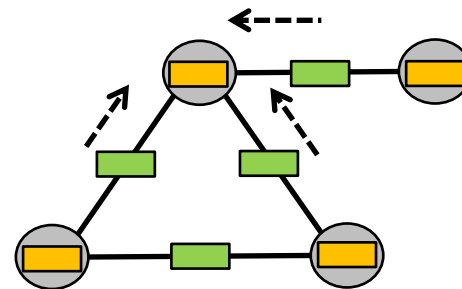
- We can divide the propagation process in two steps: 'node to edge' and 'edge to node' updates.



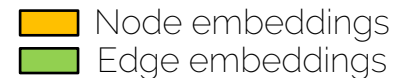
Initial Graph



'Node to edge' Update



'Edge to Node' Update



'Node to edge' updates

- At every message passing step l , first do:

$$h_{(i,j)}^{(l)} = \mathcal{N}_e \left([h_i^{(l-1)}, h_{(i,j)}^{(l-1)}, h_j^{(l-1)}] \right)$$

Embedding of node i in
the previous message
passing step

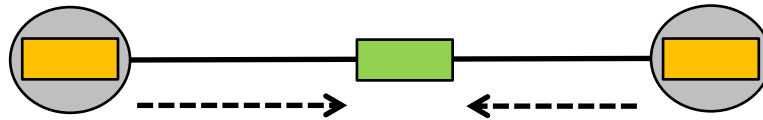
Embedding of
edge (i,j) in the
previous message
passing step

Embedding of node
 j in the previous
message passing
step

'Node to edge' updates

- At every message passing step l , first do:

$$h_{(i,j)}^{(l)} = \mathcal{N}_e \left([h_i^{(l-1)}, h_{(i,j)}^{(l-1)}, h_j^{(l-1)}] \right)$$

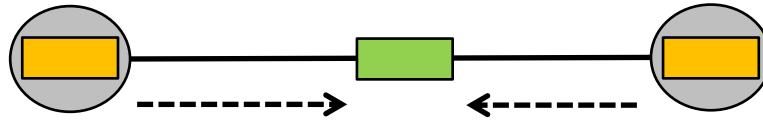


'Node to edge' updates

- At every message passing step l , first do:

$$h_{(i,j)}^{(l)} = \underbrace{\mathcal{N}_e}_{\text{Learnable function}} \left([h_i^{(l-1)}, h_{(i,j)}^{(l-1)}, h_j^{(l-1)}] \right)$$

Learnable function (e.g. MLP) with shared weights across the entire graph



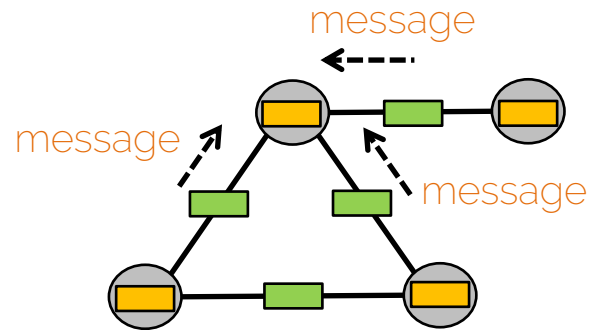
'Edge to node' updates

- After a round of edge updates, each edge embedding contains information about its pair of incident nodes
- Then, edge embeddings are used to update nodes:

$$m_i^{(l)} = \underbrace{\Phi}_{\text{Order invariant operation}} \left(\left\{ h_{(i,j)}^{(l)} \right\}_{j \in \underbrace{N_i}_{\text{Neighbors of node } i}} \right)$$

Order invariant operation (e.g. sum, mean, max)

Neighbors of node i



'Edge to node' updates

- After a round of edge updates, each edge embedding contains information about its pair of incident nodes
- Then, edge embeddings are used to update nodes:

$$m_i^{(l)} = \Phi \left(\left\{ h_{(i,j)}^{(l)} \right\}_{j \in N_i} \right)$$
$$h_i^{(l)} = \underbrace{\mathcal{N}_v}_{\text{Learnable function}} \left(\left[m_i^{(l)}, h_i^{(l-1)} \right] \right)$$

Learnable function (e.g. MLP) with shared weights across the entire graph

The aggregation provides each node embedding with contextual information about its neighbors

Remarks

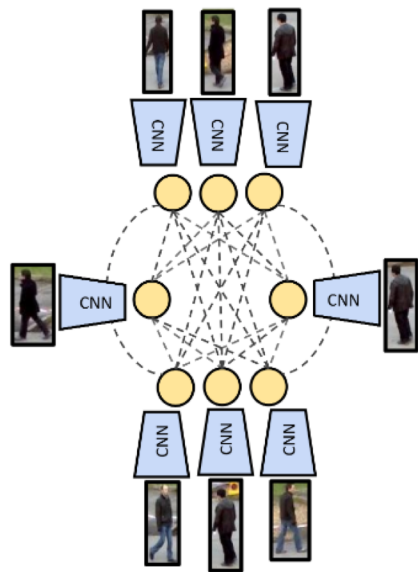
- **Main goal:** obtaining node and edge embeddings that contain *context information* encoding graph topology and neighbor's feature information.
- After repeating the node and edge updates for l steps, each node (resp. edge) embedding contains information about all nodes (resp. edge) at distance l (resp. $l - 1$) → Think of iterations as layers in a CNN
- Observe that all operations used are differentiable, hence, MPNs can be used within end-to-end pipelines
- There is vast literature on different instantiations, as well as variations of the MPN framework we presented. See Battaglia et al. for an extensive review.

MOT with Message Passing Networks

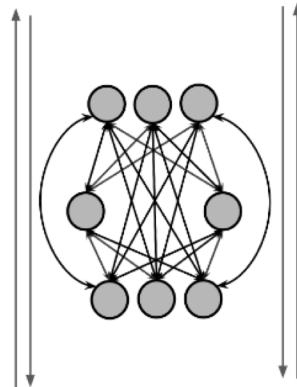
MOT with MPN: Overview



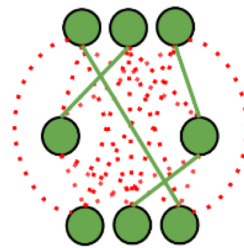
(a) Input



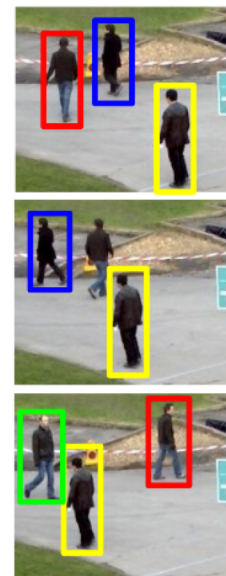
(b) Graph Construction + Feature En-
coding



(c) Neural Message Passing



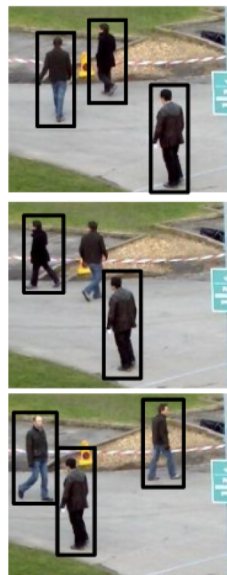
(d) Edge Classification



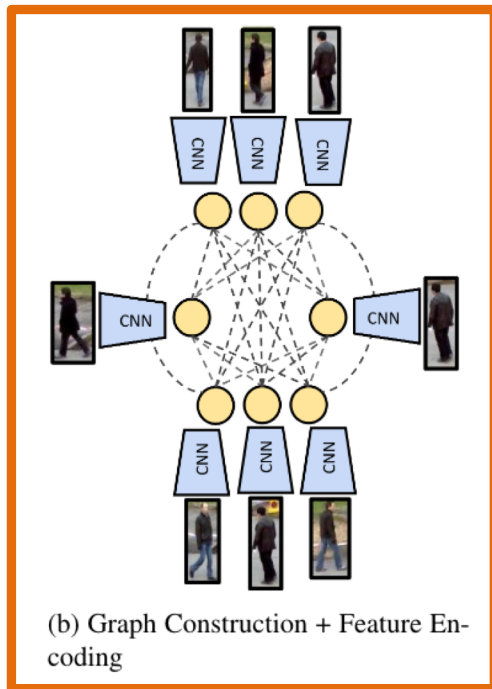
(e) Output

MOT with MPN: Overview

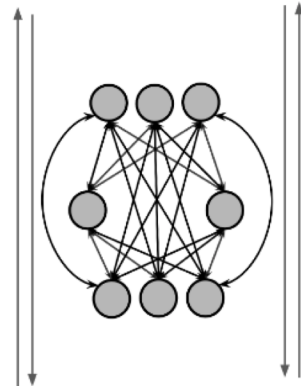
Encode appearance and scene geometry cues into node and edge embeddings



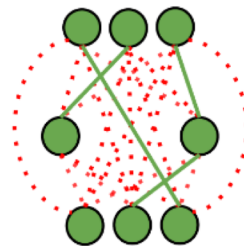
(a) Input



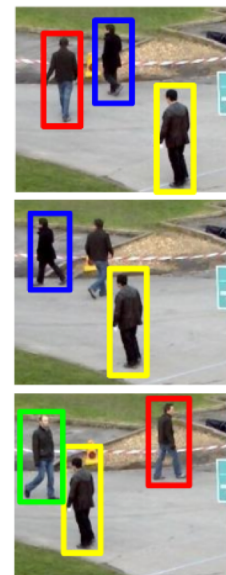
(b) Graph Construction + Feature Encoding



(c) Neural Message Passing



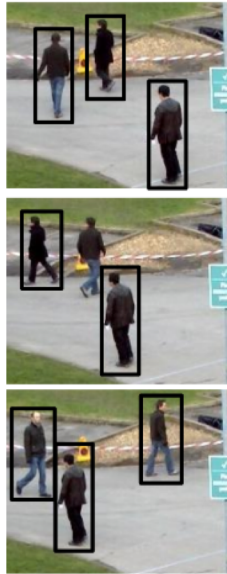
(d) Edge Classification



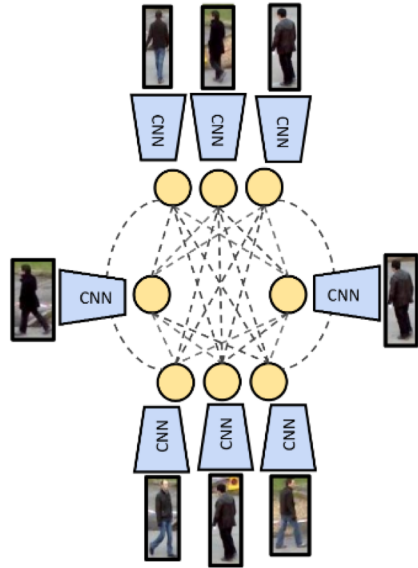
(e) Output

MOT with MPN: Overview

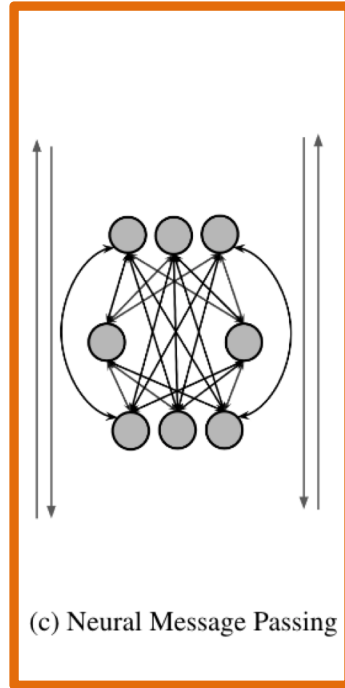
Propagate cues across the entire graph with neural message passing



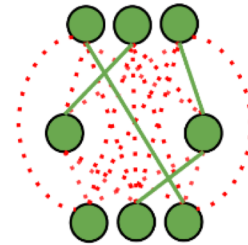
(a) Input



(b) Graph Construction + Feature En-
coding



(c) Neural Message Passing



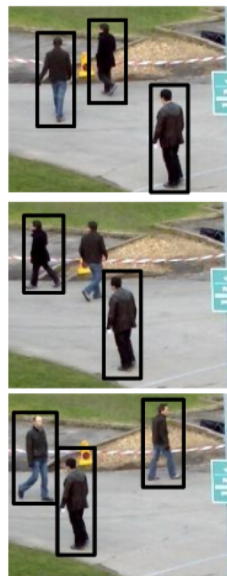
(d) Edge Classification



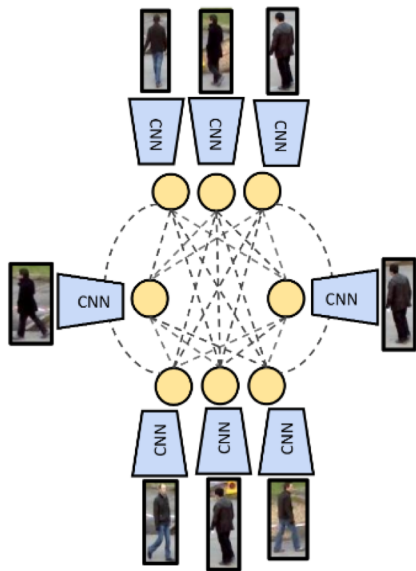
(e) Output

MOT with MPN: Overview

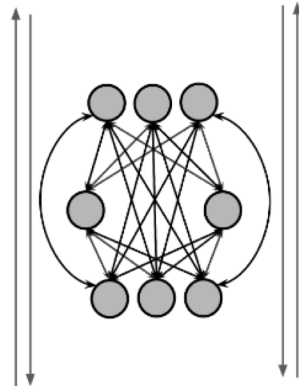
Learn to directly predict solutions of the tracking graph problem by classifying edge embeddings



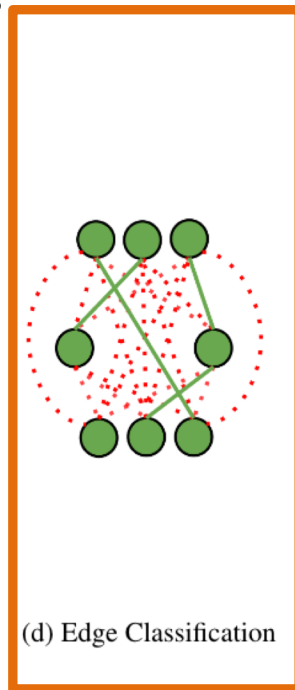
(a) Input



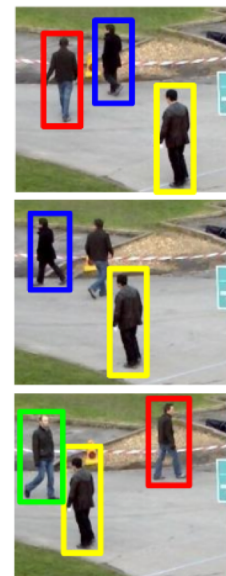
(b) Graph Construction + Feature Encoding



(c) Neural Message Passing



(d) Edge Classification

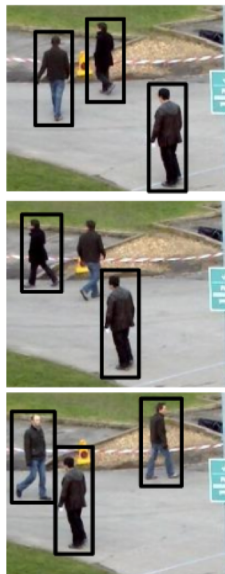


(e) Output

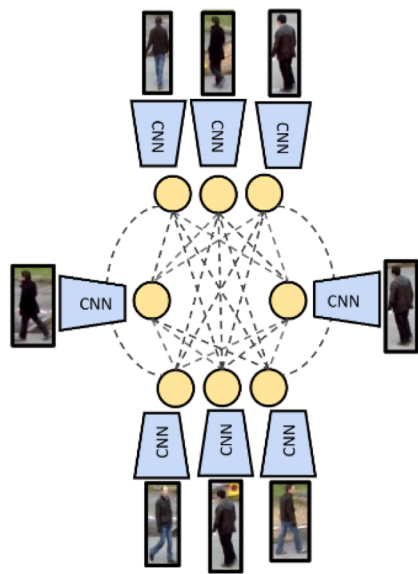
MOT with MPN: Overview

Feature Extraction

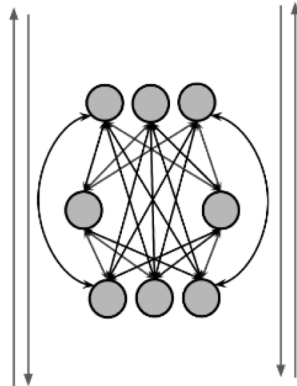
Learnable Data Association



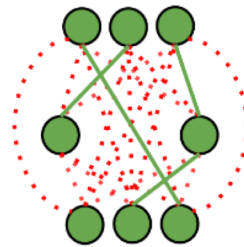
(a) Input



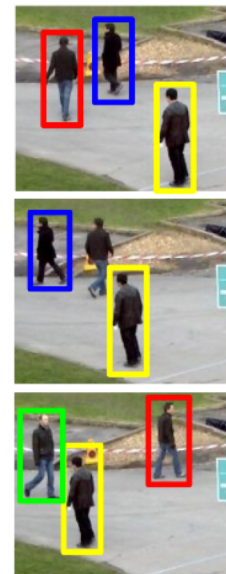
(b) Graph Construction + Feature Encoding



(c) Neural Message Passing



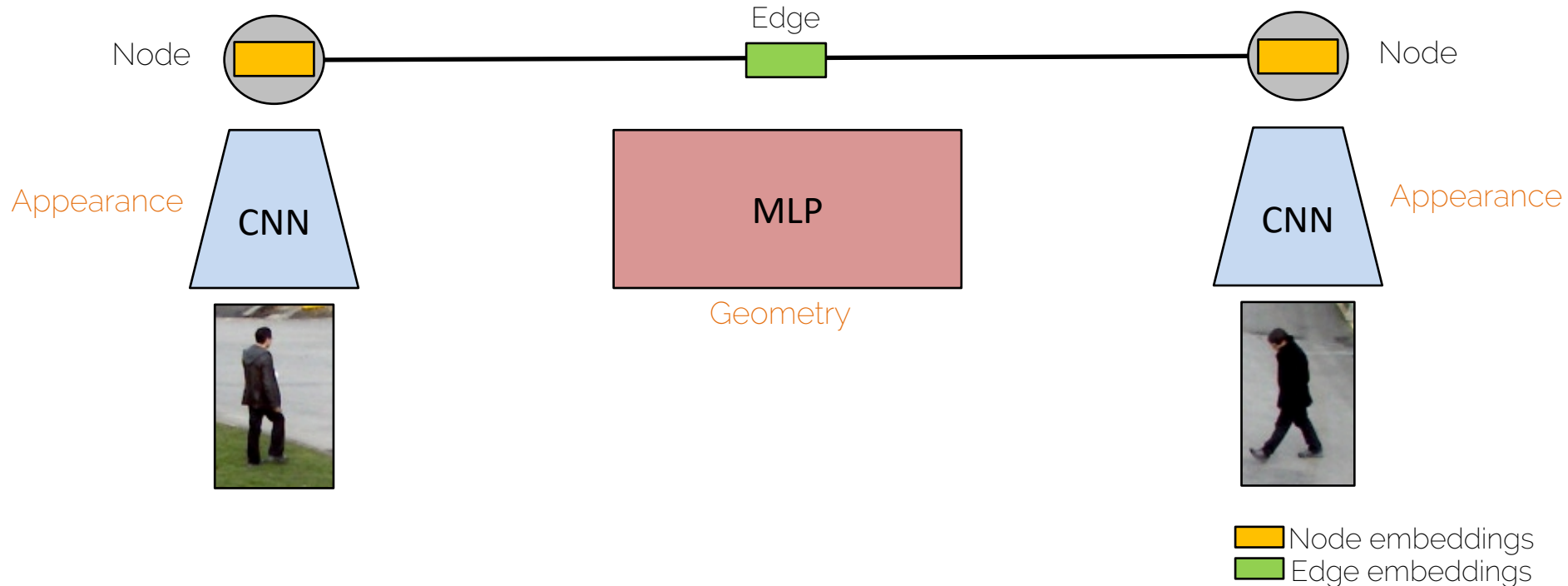
(d) Edge Classification



(e) Output

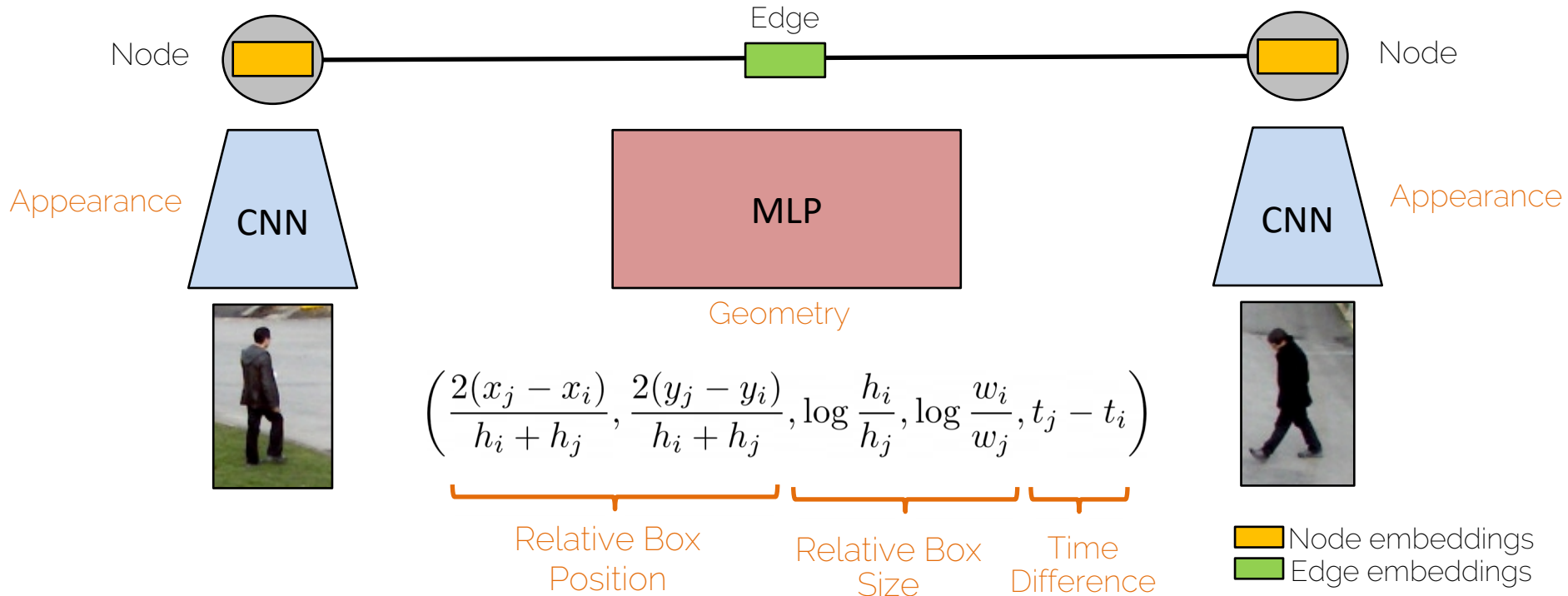
Feature encoding

- Appearance and geometry encodings



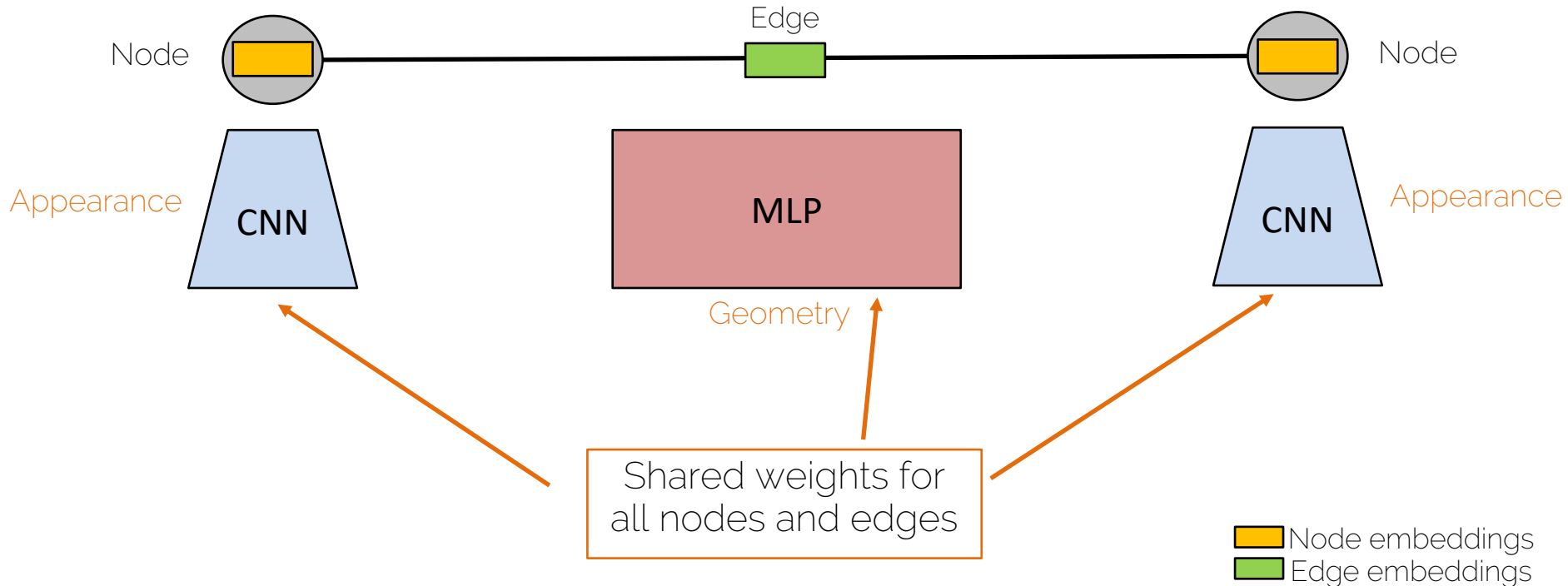
Feature encoding

- Appearance and geometry encodings



Feature encoding

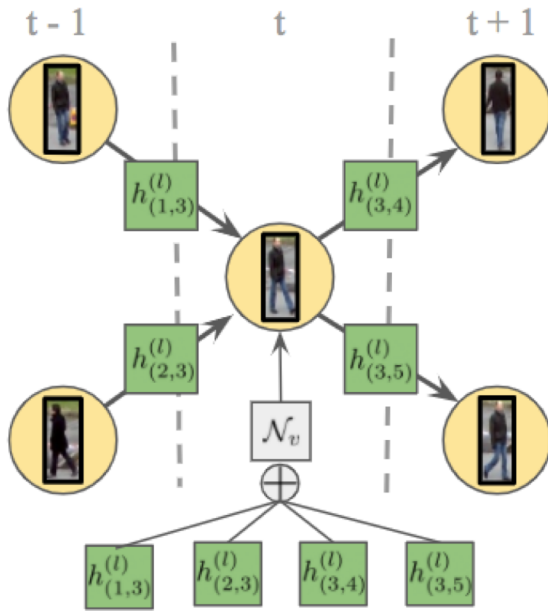
- Appearance and geometry encodings



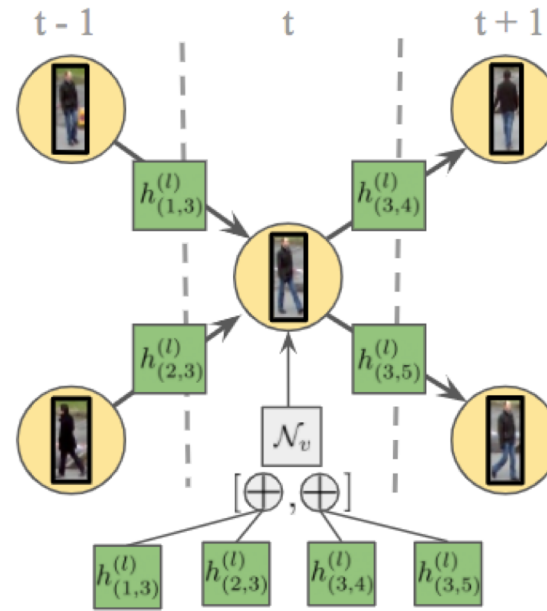
Feature encoding

- **Goal:** propagate these embeddings across the entire graph in order to obtain new embeddings encoding high-order information among detections

Time-aware Message Passing



All edge embeddings are aggregated at once



Aggregation of edge embeddings is separated between past / future frames

Classifying edges

- After several iterations of message passing, each edge embedding contains high-order information about other detections
- We feed the embeddings to an MLP that predicts whether an edge is active/inactive

$$\mathcal{L} = \frac{-1}{|E|} \sum_{l=l_0}^{l=L} \sum_{(i,j) \in E} w \cdot y_{(i,j)} \log(\hat{y}_{(i,j)}^{(l)}) + (1 - y_{(i,j)}) \log(1 - \hat{y}_{(i,j)}^{(l)})$$

Sum over the last steps

Weight to balance active / inactive edges

Edge predictions (w. sigmoid) at iteration l

Binary cross-entropy

Obtaining final solutions

- After classifying edges, we get a prediction between 0 and 1 for each edge in the graph.
- To obtain the final edge values (with value either 0 or 1) we can:
 - Use a simple rounding scheme and greedily check constraint satisfaction (remember Linear Program constraints)
 - We can use another LP to project our solution to an integer solution

Some results

- With time-aware updates, around 98% of constraints are automatically satisfied, and rounding takes negligible time.
- The overall method is fast (~6 fps) and achieves SOTA in the MOT Challenge by a significant margin

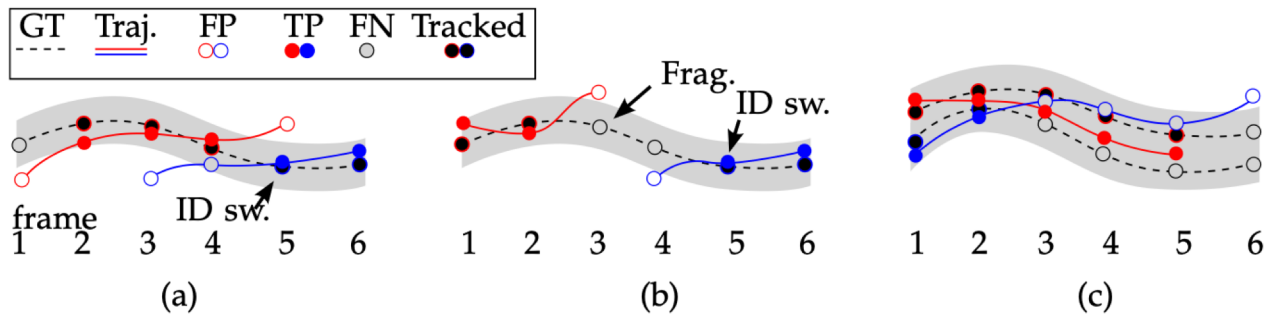
MOT evaluation

Evaluation metrics

- Compute a set of measures per frame
 - Perform matching between predictions and ground truth (we will use exactly the same Hungarian algorithm)
 - FP = False positives
 - FN = False negatives (missing detections)
 - IDsw: identity switches

Evaluation metrics

- How do we compute ID switches?



- (a) An ID switch is counted because the ground truth track is assigned first to red, then to blue.
- (b) You count both an ID switch (red and blue both assigned to the same ground truth), but also a fragmentation (Frag.) because the ground truth coverage was cut.
- (c) Identity is preserved. If two trajectories overlap with a ground truth trajectory (within a threshold), the one that forces least ID switches is chosen (the red one).

Evaluation metrics

- Compute a set of measures per frame
 - Perform matching between predictions and ground truth (we will use exactly the same Hungarian algorithm)
 - FP = False positives
 - FN = False negatives (missing detections)
 - IDsw: identity switches

Multi-object tracking accuracy \longrightarrow
$$\text{MOTA} = 1 - \frac{\sum_t (\text{FN}_t + \text{FP}_t + \text{IDSW}_t)}{\sum_t \text{GT}_t},$$
 \longleftarrow Ground truth

Datasets

- MOTChallenge: www.motchallenge.net (people)
 - Several challenges from less to more crowded



- KITTI benchmark: <http://www.cvlibs.net/datasets/kitti/> (vehicles)
- UA-Detrac: <http://detrac-db.rit.albany.edu> (vehicles)

Multiple object tracking