Learning a Neural Solver for Multiple Object Tracking

Guillem Brasó* Laura Leal-Taixé

Technical University of Munich

Abstract

Graphs offer a natural way to formulate Multiple Object Tracking (MOT) within the tracking-by-detection paradigm. However, they also introduce a major challenge for learning methods, as defining a model that can operate on such structured domain is not trivial. As a consequence, most learning-based work has been devoted to learning better features for MOT, and then using these with well-established optimization frameworks. In this work, we exploit the classical network flow formulation of MOT to define a fully differentiable framework based on Message Passing Networks (MPNs). By operating directly on the graph domain, our method can reason globally over an entire set of detections and predict final solutions. Hence, we show that learning in MOT does not need to be restricted to feature extraction, but it can also be applied to the data association step. We show a significant improvement in both MOTA and IDF1 on three publicly available benchmarks. Our code is available at https://bit.ly/motsolv.

1. Introduction

Multiple object tracking (MOT) is the task of determining the trajectories of all object instances in a video. It is a fundamental problem in computer vision, with applications such as autonomous driving, biology, and surveillance. Despite its relevance, it remains a challenging task and a relatively unexplored territory in the context of deep learning.

In recent years, *tracking-by-detection* has been the dominant paradigm among state-of-the-art methods in MOT. This two step approach consists in first obtaining frame-by-frame object detections, and then linking them to form trajectories. While the first task can be addressed with learning-based detectors [60, 1], the latter, data association, is generally formulated as a graph partitioning problem [75, 84, 86, 48, 8]. In this graph view of MOT, a node represents an object detection, and an edge represents the connection between two nodes. An active edge indicates the two detections belong to the same trajectory. Solving the graph partitioning task, i.e., finding the set of active edges or trajectories, can also be decomposed into two stages. First, a cost is assigned to each edge in the graph encoding the likelihood of two detections belonging to the same trajectory. After that, these costs are used within a graph optimization framework to obtain the optimal graph partition.

Previous works on graph-based MOT broadly fall into two categories: those that focus on the graph formulation, and those that focus on learning better costs. In the first group, numerous research has been devoted to establishing complex graph optimization frameworks that combine several sources of information, with the goal of encoding high-order dependencies between detections [73, 34, 30, 31]. Such approaches often use costs that are *handcrafted* to some extent. In the second group, several works adopt a simpler and easier to optimize graph structure, and focus instead on improving edge cost definition by leveraging deep learning techniques [44, 70, 67, 89, 80]. By exploiting siamese convolutional neural networks (CNN), these approaches can encode reliable pairwise interactions among objects, but fail to account for high-order information in the scene. Overall, these two lines of work present a dilemma: should MOT methods focus on improving the graph optimization framework or the feature extraction?

We propose to combine both tasks into a unified learning-based solver that can: (i) learn features for MOT, and (ii) learn to provide a solution by reasoning over the entire graph. To do so, we exploit the classical network flow formulation of MOT [87] to define our model. Instead of learning pairwise costs and then using these within an available solver, our method learns to directly predict final partitions of the graph into trajectories. Towards this end, we perform learning directly in the natural MOT domain, i.e., in the graph domain, with a message passing network (MPN). Our MPN learns to combine deep features into high-order information across the graph. Hence, our method is able to account for global interactions among detections despite relying on a simple graph formulation. We show that our framework yields substantial improvements with respect to state of the art, without requiring heavily engineered features and being up to one order of magnitude faster than some traditional graph partitioning methods.

*Correspondence to: guillem.braso@tum.de.

To summarize, we make the following **contributions**:

- We propose a MOT solver based on message passing networks, which can exploit the natural graph structure of the problem to perform both feature learning as well as final solution prediction.
- We propose a novel time-aware neural message passing update step inspired by classic graph formulations of MOT.
- We show significantly improved state-of-the-art results of our method in three public benchmarks.

2. Related Work

Most state-of-the-art MOT works follow the tracking-by-detection paradigm which divides the problem into two steps: (i) detecting pedestrian locations independently in each frame, for which neural networks are currently the state-of-the-art [61, 1, 82], and (ii) linking corresponding detections across time to form trajectories.

Tracking as a Graph Problem. Data association can be done on a frame-by-frame basis for online applications [10, 22, 58] or track-by-track [7]. For video analysis tasks that can be done offline, batch methods are preferred since they are more robust to occlusions. The standard way to model data association is by using a graph, where each detection is a node, and edges indicates possible link among them. The data association can then be formulated as maximum flow [8] or, equivalently, minimum cost problem with either fixed costs based on distance [32, 59, 86], including motion models [48], or learned costs [45]. Both formulations can be solved optimally and efficiently. Alternative formulations typically lead to more involved optimization problems, including minimum cliques [85], general-purpose solvers, e.g., multi-cuts [75]. A recent trend is to design ever more complex models which include other vision input such as reconstruction for multi-camera sequences [49, 79], activity recognition [17], segmentation [55], keypoint trajectories [15] or joint detection [75].

Learning in Tracking. It is no secret that neural networks are now dominating the state-of-the-art in many vision tasks since [41] showed their potential for image classification. The trend has also arrived in the tracking community, where learning has been used primarily to learn a mapping from image to optimal costs for the aforementioned graph algorithms. The authors of [42] use a siamese network to directly learn the costs between a pair of detections, while a mixture of CNNs and recurrent neural networks (RNN) is used for the same purpose in [64]. More evolved quadruplet networks [70] or attention networks [89] have lead to improved results. In [63], authors showed the importance of learned reID features for multi-object tracking. All aforementioned methods learn the costs independently from the optimization method that actually computes the final trajectories. In contrast, [38, 76, 67] incorporate the optimization solvers into learning. The main idea behind these methods is that costs also need to be optimized for the solver in which they will be used. [38, 76, 24] rely on structured learning losses while [67] proposes a more general bi-level optimization framework. These works can be seen as similar to ours in spirit, given our common goal of incorporating the full inference model into learning for MOT. However, we follow a different approach towards this end: we propose to directly learn a solver and treat data association as a classification task, while their goal is to adapt their methods to perform well with non-learnable solvers. Moreover, all these works are limited to learning either pairwise costs [24, 67] or additional quadratic terms [76, 38] but cannot incorporate higher-order information as our method. Instead, we propose to leverage the common graph formulation of MOT as a domain in which to perform learning.

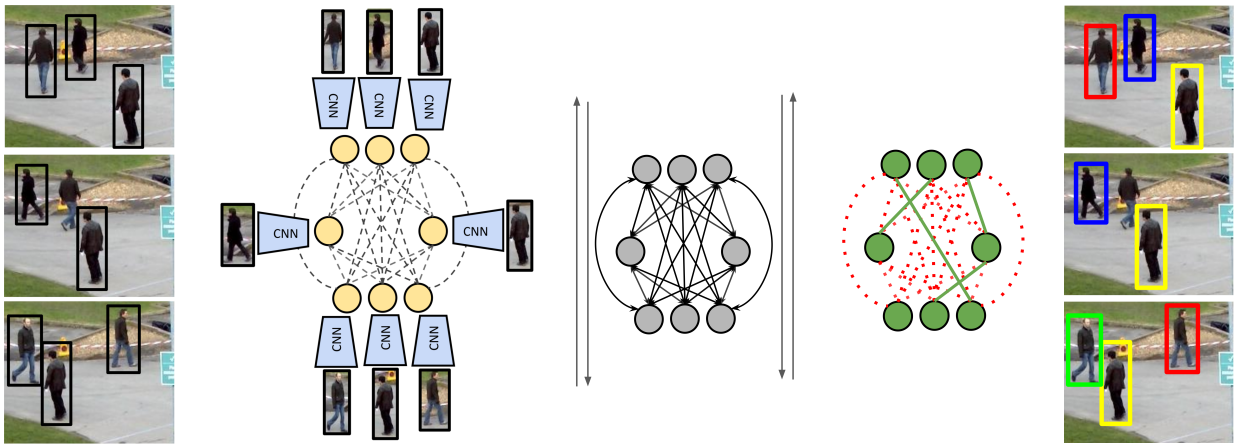
Deep Learning on Graphs. Graph Neural Networks (GNNs) were first introduced in [66] as a generalization of neural networks that can operate on graph-structured domains. Since then, several works have focused on further developing and extending them by developing convolutional variants [11, 20, 40]. More recently, most methods were encompassed within a more general framework termed neural message passing [26] and further extended in [5] as graph networks. Given a graph with some initial features for nodes and optionally edges, the main idea behind these models is to embed nodes (and edges) into representations that take into account not only the node's own features but also those of its neighbors in the graph, as well as the graph overall topology. These methods have shown remarkable performance at a wide variety of areas, ranging from chemistry [26] to combinatorial optimization [51]. Within vision, they have been successfully applied to problems such as human action recognition [27], visual question answering [56] or single object tracking [25].

3. Tracking as a Graph Problem

Our method's formulation is based on the classical min-cost flow view of MOT [87]. In order to provide some background and formally introduce our approach, we start by providing an overview of the network flow MOT formulation. We then explain how to leverage this framework to reformulate the data association task as a learning problem.

3.1. Problem Statement

In tracking-by-detection, we are given as input a set of object detections $\mathcal{O} = \{o_1, \dots, o_n\}$, where n is the total number of objects for all frames of a video. Each detection is represented by $o_i = (a_i, p_i, t_i)$, where a_i denotes the raw pixels of the bounding box, p_i contains its 2D image coordinates and t_i its timestamp. A trajectory is defined as a



(a) Input (b) Graph Construction + Feature Encoding (c) Neural Message Passing (d) Edge Classification (e) Output

Figure 1: Overview of our method. (a) We receive as input a set of frames and detections. (b) We construct a graph in which nodes represent detections, and all nodes at different frames are connected by an edge. (c) We initialize node embeddings in the graph with a CNN, and edge embeddings with an MLP encoding geometry information (not shown in figure). (c) The information contained in these embeddings is propagated across the graph for a fixed number of iterations through neural message passing. (d) Once this process terminates, the embeddings resulting from neural message passing are used to classify edges into active (colored with green) and non-active (colored with red). During training, we compute the cross-entropy loss of our predictions w.r.t. ground truth labels. (e) At inference, we follow a simple rounding scheme to binarize our classification scores and obtain final trajectories.

set of time-ordered object detections $T_i = \{o_{i_1}, \dots, o_{i_{n_i}}\}$, where n_i is the number of detections that form trajectory i. The goal of MOT is to find the set of trajectories $\mathcal{T}_* = \{T_1, \dots, T_m\}$, that best explains the observations $\mathcal{O}$.

The problem can be modelled with an undirected graph $G = (V, E)$, where $V := \{1, \dots, n\}$, $E \subset V \times V$, and each node $i \in V$ represents a unique detection $o_i \in \mathcal{O}$. The set of edges E is constructed so that every pair of detections, i.e., nodes, in different frames is connected, hence allowing to recover trajectories with missed detections. Now, the task of dividing the set of original detections into trajectories can be viewed as grouping nodes in this graph into disconnected components. Thus, each trajectory $T_i = \{o_{i_1}, \dots, o_{i_{n_i}}\}$ in the scene can be mapped into a group of nodes $\{i_1, \dots, i_{n_i}\}$ in the graph and vice-versa.

3.2. Network Flow Formulation

In order to represent graph partitions, we introduce a binary variable for each edge in the graph. In the classical minimum cost flow formulation[1] [87], this label is defined to be 1 between edges connecting nodes that (i) belong to the same trajectory, and (ii) are temporally consecutive inside a trajectory; and 0 for all remaining edges.

A trajectory $T_i = \{o_{i_1}, \dots, o_{i_{n_i}}\}$ is equivalently denoted by the set of edges $\{(i_1, i_2), \dots, (i_{n_i - 1}, i_{n_i})\} \subset E$, corresponding to its time-ordered path in the graph. We will use this observation to formally define the edge labels. For every pair of nodes in different timestamps, $(i, j) \in E$, we define a binary variable $y_{(i,j)}$ as:

$$y_{(i,j)} := \begin{cases} 1 & \exists T_k \in \mathcal{T}_* \text{ s.t. } (i,j) \in T_k \\ 0 & \text{otherwise.} \end{cases}$$

An edge (i, j) is said to be *active* whenever $y_{(i,j)} = 1$. We assume trajectories in $\mathcal{T}$ to be node-disjoint, i.e., a node cannot belong to more than one trajectory. Therefore, $\hat{y}$ must satisfy a set of linear constraints. For each node $i \in V$:

$$\sum_{(j,i) \in E \text{ s.t. } t_i > t_j} y_{(j,i)} \leq 1 \tag{1}$$

$$\sum_{(i,k) \in E \text{ s.t. } t_i < t_k} y_{(i,k)} \leq 1 \tag{2}$$

These inequalities are a simplified version of the *flow conservation constraints* [2]. In our setting, they enforce that every node gets linked via an active edge to, at most, one node in past frames and one node in upcoming frames.

[1] We present a simplified version of the minimum cost flow MOT formulation [87]. Specifically, we omit both sink and source nodes (and hence their corresponding edges) and we assume detection edges to be constant and 1-valued. We provide further details on our simplification and its relationship to the original problem in Appendix A.

3.3. From Learning Costs to Predicting Solutions

In order to obtain a graph partition with the framework we have described, the standard approach is to first associate a cost $c_{(i,j)}$ to each binary variable $y_{(i,j)}$. This cost encodes the likelihood of the edge being active [43, 42, 67]. The final partition is found by optimizing:

$$\begin{aligned} \min_y \quad & \sum_{(i,j)\in E} c_{(i,j)} y_{(i,j)} \\ \text{Subject to:} \quad & \text{Equation } (1) \\ & \text{Equation } (2) \\ & y_{(i,j)} \in \{0,1\}, \qquad (i,j) \in E \end{aligned}$$

which can be solved with available solvers in polynomial time [6, 3].

We propose to, instead, directly learn to predict which edges in the graph will be active, i.e., predict the final value of the binary variable y. To do so, we treat the task as a classification problem over edges, where our labels are the binary variables y. Overall, we exploit the classical network flow formulation we have just presented to treat the MOT problem as a fully learnable task.

4. Learning to Track with Message Passing Networks

Our main contribution is a differentiable framework to train multi-object trackers as edge classifiers, based on the graph formulation we described in the previous section. Given a set of input detections, our model is trained to predict the values of the binary *flow* variables y for every edge in the graph. Our method is based on a novel message passing network (MPN) able to capture the graph structure of the MOT problem. Within our proposed MPN framework, appearance and geometry cues are propagated across the entire set of detections, allowing our model to reason globally about the entire graph.

Our pipeline is composed of four main stages:

1. Graph Construction: Given a set of object detections in a video, we construct a graph where nodes correspond to detections and edges correspond to connections between nodes (Section 3.2).

2. Feature Encoding: We initialize the node *appearance* feature embeddings from a convolutional neural network (CNN) applied on the bounding box image. For each edge, i.e., for every pair of detections in different frames, we compute a vector with features encoding their bounding box relative size, position and time distance. We then feed it to a multi-layer perceptron (MLP) that returns a *geometry* embedding (Section 4.3).

3. Neural Message Passing: We perform a series of message passing steps over the graph. Intuitively, for each round of message passing, nodes share appearance information with their connecting edges, and edges share geometric information with their incident nodes. This yields updated embeddings for node and edges containing *higher-order* information that depends on the overall graph structure (Section 4.1 and 4.2).

4. Training: We use the final edge embeddings to perform binary classification into active/non-active edges, and train our model using the cross-entropy loss (Section 4.4).

At test time, we use our model's prediction per edge as a continuous approximation (between 0 and 1) of the target *flow* variables. We then follow a simple scheme to round them, and obtain the final trajectories.

For a visual overview of our pipeline, see Figure 1.

4.1. Message Passing Networks

In this section, we provide a brief introduction to MPNs based on the work presented in [26, 39, 4, 5]. Let $G = (V, E)$ be a graph. Let $h_i^{(0)}$ be a node embedding for every $i \in V$, and $h_{(i,j)}^{(0)}$ an edge embedding for every $(i,j) \in E$. The goal of MPNs is to learn a function to propagate the information contained in nodes and edge feature vectors across G.

The propagation procedure is organized in embedding updates for edges and nodes, which are known as *message passing steps* [26]. In [5, 39, 4], each message passing step is divided, in turn, into two updates: one from from nodes to edges $(v \to e)$, and one from edges to nodes $(e \to v)$. The updates are performed sequentially for a fixed number of iterations L. For each $l \in \{1, \dots, L\}$, the general form of the updates is the following [5]:

$$(v \to e) \quad h_{(i,j)}^{(l)} = \mathcal{N}_e \left([h_i^{(l-1)}, h_j^{(l-1)}, h_{(i,j)}^{(l-1)}] \right) \tag{3}$$

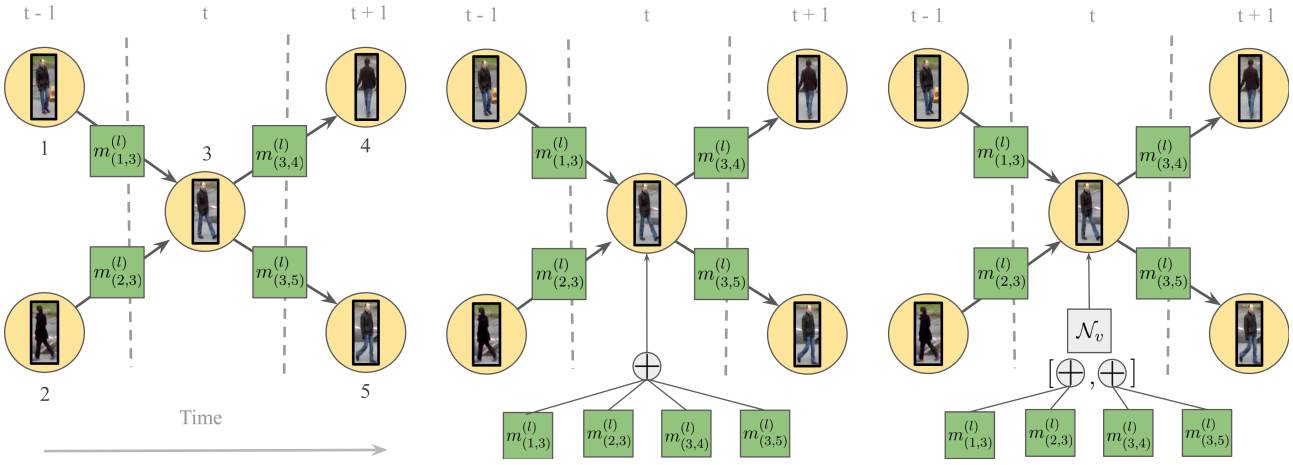
$$(e \to v) \quad m_{(i,j)}^{(l)} = \mathcal{N}_v \left([h_i^{(l-1)}, h_{(i,j)}^{(l)}] \right) \tag{4}$$

$$h_i^{(l)} = \Phi \left(\left\{ m_{(i,j)}^{(l)} \right\}_{j \in N_i} \right) \tag{5}$$

Where $\mathcal{N}_e$ and $\mathcal{N}_v$ represent learnable functions, e.g., MLPs, that are shared across the entire graph. $[.]$ denotes concatenation, $N_i \subset V$ is the set of adjacent nodes to i, and Φ denotes an order-invariant operation, e.g., a summation, maximum or an average. Note, after L iterations, each node contains information of all other nodes at distance L in the graph. Hence, L plays an analogous role to the receptive field of CNNs, allowing embeddings to capture context information.

4.2. Time-Aware Message Passing

The previous message passing framework was designed to work on arbitrary graphs. However, MOT graphs have a very specific structure that we propose to exploit. Our goal is to encode a MOT-specific inductive bias in our network, specifically, in the node update step.



(a) Initial Setting (b) Vanilla node update (c) Time-aware node update

Figure 2: Visualization of node updates during message passing. Arrow directions in edges show time direction. Note the time division in $t-1$, t, and $t+1$. In this case, we have $N_3^{past} = \{1, 2\}$ and $N_3^{fut} = \{4, 5\}$. 2a shows the starting point after an edge update has been performed (equation 3), and the intermediate node update embeddings (equation 4) have been computed. 2b shows the standard node update in vanilla MPNs, in which all neighbors' embeddings are aggregated jointly. 2c shows our proposed update, in which embeddings from past and future frames are aggregated separately, then concatenated and fed into an MLP to obtain the new node embedding.

Recall the node update depicted in Equations 4 and 5, which allows each node to be compared with its neighbors and aggregate information from all of them to update its embedding with further context. Recall also the structure of our flow conservation constraints (Equations 1 and 2), which imply that each node can be connected to, at most, one node in future frames and another one in past frames. Arguably, aggregating all neighboring embeddings at once makes it difficult for the updated node embedding to capture whether these constraints are being violated or not (see Section 5.2 for constraint satisfaction analysis).

More generally, explicitly *encoding* the temporal structure of MOT graphs into our MPN formulation can be a useful prior for our learning task. Towards this goal, we modify Equations 4 and 5 into time-aware update rules by dissecting the aggregation into two parts: one over nodes in the past, and another over nodes in the future. Formally, let us denote the neighboring nodes of i in future and past frames by N_i^{fut} and N_i^{past}, respectively. Let us also define two different MLPs, namely, $\mathcal{N}_v^{fut}$ and $\mathcal{N}_v^{past}$. At each message passing step l and for every node $i \in V$, we start by computing *past* and *future* edge-to-node embeddings for all of its neighbors $j \in N_i$ as:

$$m^{(l)}_{(i,j)} = \begin{cases} \mathcal{N}_v^{past}\left([h_i^{(l-1)}, h^{(l)}_{(i,j)}, h^{(0)}_{(i)}]\right) & \text{if} \quad j \in N_i^{past} \\ \mathcal{N}_v^{fut}\left([h_i^{(l-1)}, h^{(l)}_{(i,j)}, h^{(0)}_{(i)}]\right) & \text{if} \quad j \in N_i^{fut} \end{cases} \tag{6}$$

Note, the initial embeddings $h^{(0)}_{(i)}$ have been added to the computation[2]. After that, we aggregate these embeddings separately, depending on whether they were in future or past positions with respect to i:

$$h^{(l)}_{i,past} = \sum_{j \in N_i^{past}} m^{(l)}_{(i,j)} \tag{7}$$

$$h^{(l)}_{i,fut} = \sum_{j \in N_i^{fut}} m^{(l)}_{(i,j)} \tag{8}$$

Now, these operations yield *past* and *future* embeddings $h^{(l)}_{i,past}$ and $h^{(l)}_{i,fut}$, respectively. We compute the final updated node embedding by concatenating them and feeding the result to one last MLP, denoted as $\mathcal{N}_v$:

$$h_i^{(l)} = \mathcal{N}_v([h^{(l)}_{i,past}, h^{(l)}_{i,fut}]) \tag{9}$$

We summarize our time-aware update in Figure 2c. As we demonstrate experimentally (see Section 5.2), this simple architectural design results in a significant performance improvement with respect to the *vanilla* node update of MPNs, shown in Figure 2b.

4.3. Feature Encoding

The initial embeddings that our MPN receives as input are produced by other backpropagatable networks.

[2]This skip connection ensures that our model does not *forget* its initial features during message passing, and we apply it analogously with initial edge features in Equation 3.

Appearance Embedding. We rely on a convolutional neural network (CNN), denoted as $\mathcal{N}_v^{enc}$, to *learn* to extract a feature embeddings directly from RGB data. For every detection $o_i \in \mathcal{O}$, and its corresponding image patch a_i, we obtain o_i's corresponding node embedding by computing $h_i^{(0)} := \mathcal{N}_v^{enc}(a_i)$.

Geometry Embedding. We seek to obtain a representation that encodes, for each pair of detections in different frames, their relative position size, as well as distance in time. For every pair of detections o_i and o_j with timestamps $t_i \neq t_j$, we consider their bounding box coordinates parameterized by top left corner image coordinates, height and width, i.e., (x_i, y_i, h_i, w_i) and (x_j, y_j, h_j, w_j). We compute their relative distance and size as:

$$\left(\frac{2(x_j - x_i)}{h_i + h_j}, \frac{2(y_j - y_i)}{h_i + h_j}, \log\frac{h_i}{h_j}, \log\frac{w_i}{w_j}\right)$$

We then concatenate this coordinate-based feature vector with the time difference $t_j - t_i$ and relative appearance $\|\mathcal{N}_v^{enc}(a_j) - \mathcal{N}_v^{enc}(a_i)\|_2$ and feed it to a neural network $\mathcal{N}_e^{enc}$ in order to obtain the initial edge embedding $h_{(i,j)}^{(0)}$.

4.4. Training and Inference

Training Loss. To classify edges, we use an MLP with a sigmoid-valued single output unit, that we denote as $\mathcal{N}_e^{class}$. For every edge $(i,j) \in E$, we compute our prediction $\hat{y}_{(i,j)}^{(l)}$ by feeding the output embeddings of our MPN at a given message passing step l, namely $h_{(i,j)}^{(l)}$, to $\mathcal{N}_e^{class}$. For training, we use the binary cross-entropy of our predictions over the embeddings produced in the last message passing steps, with respect to the target flow variables y:

$$\mathcal{L} = \frac{-1}{|E|}\sum_{l=l_0}^{l=L}\sum_{(i,j)\in E} w \cdot y_{(i,j)} \log(\hat{y}_{(i,j)}^{(l)}) + (1 - y_{(i,j)}) \log(1 - \hat{y}_{(i,j)}^{(l)}) \tag{10}$$

where $l_0 \in \{1, \ldots, L\}$ is the first message passing step at which predictions are computed, and w denotes a positive scalar used to weight 1-valued labels to account for the high imbalance between active and inactive edges.

Inference. During inference, we interpret the set of output values obtained from our model at the last message passing step as the solution to our MOT problem, i.e., the final value for the indicator variables y. Since these predictions are the output of a sigmoid unit, their values are between 0 and 1. An easy way to obtain hard 0 or 1 decisions is to binarize the output by thresholding. However, this procedure does not generally guarantee that the flow conservation constraints in Equations 1 and 2 are preserved. In practice, thanks to the proposed time-aware update step, our method will satisfy over 98% of the constraints on average when thresholding at 0.5. After that, a simple greedy rounding scheme suffices to obtain a feasible binary output. The exact optimal rounding solution can also be obtained efficiently with a simple linear program. We explain both procedures in Appendix B.

5. Experiments

In this section, we first present an ablation study to better understand the behavior of our model. We then compare to published methods on three datasets, and show state-of-the-art results. All experiments are done on the MOTChallenge pedestrian benchmark.

Datasets and Evaluation Metrics. The multiple object tracking benchmark MOTChallenge [3] consists of several challenging pedestrian tracking sequences, with frequent occlusions and crowded scenes. The challenge includes three separate tracking benchmarks, namely *2D MOT 2015* [46], *MOT16* [54] and *MOT17* [54]. They contain sequences with varying viewing angle, size and number of objects, camera motion and frame rate. For all challenges, we use the detections provided by MOTChallenge to ensure a fair comparison with other methods. The benchmark provides several evaluation metrics. The Multiple Object Tracking Accuracy (MOTA) [33] and ID F1 Score (IDF1) [62] are the most important ones, as they quantify two of the main aspects of multiple object tracking, namely, object coverage and identity preservation.

5.1. Implementation Details

Network Models. For the network $\mathcal{N}_v^{enc}$ used to encode detections appearances (see section 4.3), we employ a ResNet50[28] architecture pretrained on ImageNet [21], followed by global average pooling and two fully-connected layers to obtain embeddings of dimension 256.

We train the network for the task of ReIdentification (ReID) jointly on three publicly available datasets: Market1501[88], CUHK03[50] and DukeMTMC[62]. Note that using external ReID datasets is a common practice among MOT methods [73, 37, 53]. Once trained, three new fully connected layers are added after the convolutional layers to reduce the embedding size of $\mathcal{N}_v^{enc}$ to 32. The rest of the encoder and classifier networks are MLPs and their exact architectures are detailed in Table 5 in the supplementary material.

Data Augmentation. To train our network, we sample batches of 8 graphs. Each graph corresponds to small clips of 15 frames sampled at 6 frames per second for static sequences, and 9 frames per second for those with a moving camera. We do data augmentation by randomly removing nodes from the graph, hence simulating missed detections, and randomly shifting bounding boxes.

Training. We have empirically observed that additional

[3] The official MOTChallenge web page is available at https://motchallenge.net.

training of the ResNet blocks provides no significant increase in performance, but carries a significantly larger computational overhead. Hence, during training, we freeze all convolutional layers and train jointly all of the remaining model components. We train for 15000 iterations with a learning rate $3 \cdot 10^{-4}$, weight decay term 10^{-4} and an Adam Optimizer with β_1 and β_2 set to 0.9 and 0.999, respectively.

Batch Processing. We process videos offline in batches of 15 frames, with 14 overlapping frames between batches to ensure that the maximum time distance between two connected nodes in the graph remains stable along the whole graph. We prune graphs by connecting two nodes only if both are among the top-K mutual nearest neighbors (with $K = 50$) according to the ResNet features. Each batch is solved independently by our network, and for overlapping edges between batches, we average the predictions coming from the all graph solutions before the rounding step. To fill gaps in our trajectories, we perform simple bilinear interpolation along missing frames.

Baseline. Recently, [9] has shown the potential of detectors for simple data association, establishing a new baseline for MOT. To exploit it, we preprocess all sequences by first running [9] on public detections, which allows us to be fully comparable to all methods on MOTChallenge. One key drawback of [9] is its inability to fill in gaps, nor properly recover identities through occlusions. As we will show, this is exactly where our method excels. In Appendix D, we show additional results without [9].

Runtime. We build our graph on the output of [9]. Hence, we take also its runtime into account. Our method, on its own, runs at 35fps, while [9] without the added re-ID head runs at 8fps, which gives the reported average of 6.5fps.

5.2. Ablation Study

In this section, we aim to answer three main questions towards understanding our model. Firstly, we compare the performance of our time-aware neural message passing updates with respect to the time-agnostic vanilla node update described in 4.1. Secondly, we assess the impact of the number of message passing steps in network training to the overall tracking performance. Thirdly, we investigate how different information sources, namely, appearance embeddings from our CNN and relative position information, affect different evaluation metrics.

Experimental Setup. We conduct all of our experiments with the training sequences of the MOT15 and MOT17 datasets. To evaluate our models, we split MOT17 sequences into three sets, and use these to test our models with 3-fold cross-validation. We then report the best overall MOT17 metrics obtained during validation. See Appendix C.3 for more details. In order to provide a fair comparison with configurations that show poor constraint satisfaction, we use exact rounding via a linear program in all experiments (see Section 4.4).

Arch.	MOTA ↑	IDF1 ↑	MT ↑	ML ↓	FP ↓	FN ↓	ID Sw. ↓	Constr. ↑
Vanilla	63.0	67.3	586	372	41281	119542	1022	82.1
T. aware	64.0	70.0	648	362	6169	114509	602	98.8

Table 1: We investigate how our proposed update improves tracking performance with respect to a vanilla MPN. *Vanilla* stands for a basic MPN, *T. aware* denotes our proposed time-aware update. The metric *Constr* refers to the percentage of flow conservation constraints satisfied on average over entire validation sequences.

Time-Aware Message Passing. We investigate how our proposed time-aware node update affects performance. For a fair comparison, we perform hyperparameter search for our baseline. Still, we observe a significant improvement in almost all metrics, including close to 3 points in IDF1. As we expected, our model is particularly powerful at linking detections, since it exploits neighboring information and graph structure, making the decisions more robust, and hence producing significantly less identity switches. We also report the percentage of constraints that are satisfied when directly thresholding our model's output values at 0.5. Remarkably, our method with time-aware node updates is able to produce almost completely feasible results automatically, i.e., 98.8% constraint satisfaction, while the baseline has only 82.1% satisfaction. This demonstrates its ability to capture the MOT problem structure.

Number of Message Passing Steps. Intuitively, increasing the number of message passing steps L allows each node and edge embedding to encode further context, and gives edge predictions the ability to be iteratively refined. Hence, one would expect higher values to yield better performing networks. We test this hypothesis in Figure 3 by training networks with a fixed number of message passing steps, from 0 to 18. We use the case $L = 0$ as a baseline in which we train a binary classifier on top of our initial edge embeddings, and hence, no contextual information is used. As expected, we see a clear upward tendency for both IDF-1 and MOTA. Moreover, we observe a steep increase in both metrics from zero to two message passing steps, which demonstrates that the biggest improvement is obtained when switching from pairwise to high-order features in the graph. We also note that the upwards tendency stagnates around six message passing steps, and shows no improvement after twelve message passing steps. Hence, we use $L = 12$ in our final configuration.

Effect of the Features. Our model receives two main streams of information: (i) appearance information from a CNN, and (ii) geometry features from an MLP encoding relative position between detections. We test their usefulness by experimenting with combinations of three groups of features for edges: time difference, relative position and

Edge Feats.	MOTA ↑	IDF1 ↑	MT ↑	ML ↓	FP ↓	FN ↓	ID Sw. ↓
Time	58.8	52.6	529	372	13127	122800	2962
Time+Pos	63.6	68.7	631	365	6308	115506	895
Time+Pos+CNN	64.0	70.0	648	362	6169	114509	602

Table 2: We explore combinations of three sources of information for edge features: time difference in seconds (Time), relative position features (Pos) and the Euclidean distance between CNN embeddings of the two detections (CNN).

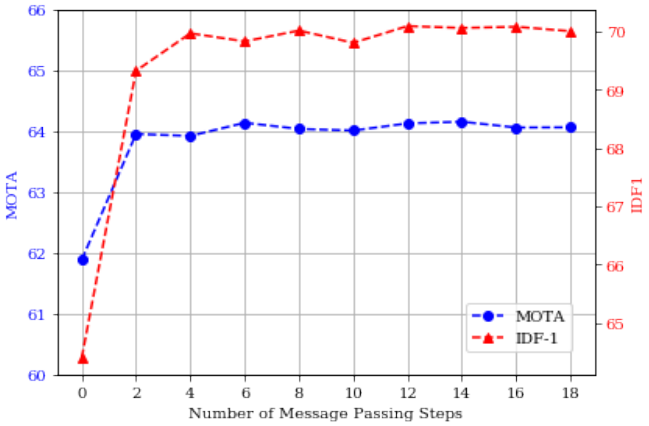


Figure 3: We report the evolution of IDF-1 and MOTA when training networks with an increasing number of message passing steps.

the euclidean distance in CNN embeddings between the two bounding boxes. Results are summarized in Table 2. We highlight the fact that relative position seems to be a key component to overall performance since its addition yields the largest relative performance increase. Nevertheless, CNN features are powerful to reduce the number of false positives and identity switches and hence, we use them in our final configuration.

5.3. Benchmark Evaluation

We report the metrics obtained by our model on the MOT15, MOT16 and MOT17 datasets in Table 3. Our method obtains state-of-the-art results by a large margin on all challenges, improving especially the IDF1 measure by 11, 6.4, and 6.6 percentage points, respectively, which demonstrates our method's strong performance in identity preservation. We attribute this performance increase to the ability of our message passing architecture to collect higher-order information. Taking into consideration neighbors' information when linking trajectories allows our method to make *globally informed* predictions, which leads inevitably to less identity switches. Moreover, we also achieve more trajectory coverage, represented by an increase in Mostly Tracked (MT) trajectories of up to 9 percentage points. It is worth noting the big performance improvement with respect

Method	MOTA ↑	IDF1 ↑	MT ↑	ML ↓	FP ↓	FN ↓	ID Sw. ↓	Hz ↑
2D MOT 2015 [46]								
Ours	**51.5**	**58.6**	**31.2**	**25.9**	7620	**21780**	**375**	6.5
Tracktor [9]	46.6	47.6	18.2	27.9	4624	26896	1290	1.8
KCF [18] (G)	38.9	44.5	16.6	31.5	7321	29501	720	0.3
AP_HWDPL_p [13] (G)	38.5	47.1	8.7	37.4	**4005**	33203	586	6.7
STRN [81]	38.1	46.6	11.5	33.4	5451	31571	1033	13.8
AMIR15 [65]	37.6	46.0	15.8	26.8	7933	29397	1026	1.9
JointMC [34] (G)	35.6	45.1	23.2	39.3	10580	28508	457	0.6
DeepFlow[67] (G)	26.8	–	–	–	–	–	–	–
MOT16 [54]								
Ours	**58.6**	**61.7**	**27.3**	**34.0**	4949	**70252**	**354**	6.5
Tracktor [9]	56.2	54.9	20.7	35.8	**2394**	76844	617	1.8
NOTA [12] (G)	49.8	55.3	17.9	37.7	7428	83614	614	–
HCC [53] (G)	49.3	50.7	17.8	39.9	5333	86795	391	0.8
LMP [74] (G)	48.8	51.3	18.2	40.1	6654	86245	481	0.5
KCF [18] (G)	48.8	47.2	15.8	38.1	5875	86567	906	0.1
GCRA [52]	48.2	48.6	12.9	41.1	5104	88586	821	2.8
FWT [29] (G)	47.8	44.3	19.1	38.2	8886	85487	852	0.6
MOT17 [54]								
Ours	**58.8**	**61.7**	**28.8**	**33.5**	17413	**213594**	**1185**	6.5
Tracktor[9]	56.3	55.1	21.1	35.3	**8866**	235449	1987	1.8
JBNOT [31] (G)	52.6	50.8	19.7	35.8	31572	232659	3050	5.4
FAMNet [19]	52.0	48.7	19.1	33.4	14138	253616	3072	–
eHAF[69] (G)	51.8	54.7	23.4	37.9	33212	236772	1834	0.7
NOTA [12] (G)	51.3	54.7	17.1	35.4	20,148	252,531	2,285	–
FWT [29] (G)	51.3	47.6	21.4	35.2	24101	247921	2648	0.2
jCC [34] (G)	51.2	54.5	20.9	37.0	25937	247822	1802	1.8

Table 3: Comparison of our method with state-of-the art. We set new state-of-the art results by a significant margin in terms of MOTA and especially IDF1. Our learned solver is more accurate while being significantly faster than most graph partitioning methods, indicated with (G).

to previous graph partitioning methods (shown as (G) in Table 3), which often use expensive optimization schemes. Not only do we surpass them by a large margin, but we are also up to one order of magnitude faster than some of them, e.g. [34]. In Appendix D, we show a more detailed comparison.

6. Conclusion

We have demonstrated how to exploit the min-cost flow formulation of MOT to treat the entire tracking problem as a learning task. We have proposed a fully differentiable pipeline in which both feature extraction and data association can be jointly learned. At the core of our algorithm lies a message passing network with a novel time-aware update step that can capture the problem's graph structure. In our experiments, we have shown a clear performance improvement of our method with respect to previous state-of-the-art. We expect our approach to open the door for future work to go beyond feature extraction for MOT, and focus, instead, on integrating learning into the overall data association task.

Acknowledgements. This research was partially funded by the Humboldt Foundation through the Sofja Kovalevskaja Award.

References

[1] J. R. ad A. Farhadi. Yolo9000: Better, faster, stronger. *CVPR*, 2017. 1, 2

[2] R. Ahuja, T. Magnanti, and J. Orlin. *Network flows: Theory, algorithms and applications*. Prentice Hall, 1993. 3

[3] R. Ahuja, T. Magnanti, and J. Orlin. *Network flows: Theory, algorithms and applications*. Prentice Hall, Upper Saddle River, NJ, USA, 1993. 4, 14

[4] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, and k. kavukcuoglu. Interaction networks for learning about objects, relations and physics. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4502–4510. Curran Associates, Inc., 2016. 4

[5] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018. 2, 4

[6] J. Berclaz, F. Fleuret, and P. Fua. Robust people tracking with global trajectory optimization. 4

[7] J. Berclaz, F. Fleuret, and P. Fua. Robust people tracking with global trajectory optimization. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 744–750, 2006. 2

[8] J. Berclaz, F. Fleuret, E. Türetken, and P. Fua. Multiple object tracking using k-shortest paths optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 33(9):1806–1819, 2011. 1, 2

[9] P. Bergmann, T. Meinhardt, and L. Leal-Taixé. Tracking without bells and whistles. *The International Conference on Computer Vision(ICCV)*, abs/1903.05625, 2019. 7, 8, 16, 17, 18

[10] M. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. van Gool. Robust tracking-by-detection using a detector confidence particle filter. *IEEE International Conference on Computer Vision (ICCV)*, pages 1515–1522, 2009. 2

[11] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *CoRR*, abs/1312.6203, 2013. 2

[12] L. Chen, H. Ai, R. Chen, and Z. Zhuang. Aggregate tracklet appearance features for multi-object tracking. *IEEE Signal Processing Letters*, 26(11):1613–1617, Nov 2019. 8, 18

[13] L. Chen, H. Ai, C. Shang, Z. Zhuang, and B. Bai. Online multi-object tracking with convolutional neural networks. pages 645–649, Sept 2017. 8

[14] L. Chen, H. Ai, Z. Zhuang, and C. Shang. Real-time multiple people tracking with deeply learned candidate selection and person re-identification. *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2018. 16

[15] W. Choi. Near-online multi-target tracking with aggregated local flow descriptor. *ICCV*, 2015. 2

[16] W. Choi. Near-online multi-target tracking with aggregated local flow descriptor. *ICCV*, 2015. 18

[17] W. Choi and S. Savarese. A unified framework for multi-target tracking and collective activity recognition. *European Conference on Computer Vision (ECCV)*, pages 215–230, 2012. 2

[18] P. Chu, H. Fan, C. C. Tan, and H. Ling. Online multi-object tracking with instance-aware tracker and dynamic model refreshment. *WACV*, abs/1902.08231, 2019. 8

[19] P. Chu and H. Ling. Famnet: Joint learning of feature, affinity and multi-dimensional assignment for online multiple object tracking. July 2019. 8

[20] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3844–3852. Curran Associates, Inc., 2016. 2

[21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 6

[22] A. Ess, B. Leibe, K. Schindler, and L. van Gool. A mobile vision system for robust multi-person tracking. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2008. 2

[23] F. Fleuret, J. Berclaz, R. Lengagne, and P. Fua. Multicamera people tracking with a probabilistic occupancy map. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):1806–1819, Feb. 2008. 14

[24] D. Frossard and R. Urtasun. End-to-end learning of multi-sensor 3d tracking by detection. *CoRR*, abs/1806.11534, 2018. 2

[25] J. Gao, T. Zhang, and C. Xu. Graph convolutional tracking. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 2

[26] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017. 2, 4

[27] M. Guo, E. Chou, D.-A. Huang, S. Song, S. Yeung, and L. Fei-Fei. Neural graph matching networks for fewshot 3d action recognition. In *The European Conference on Computer Vision (ECCV)*, September 2018. 2

[28] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 6, 15

[29] R. Henschel, L. Leal-Taixé, D. Cremers, and B. Rosenhahn. Improvements to frank-wolfe optimization for multi-detector multi-object tracking. *CVPR*, abs/1705.08314, 2017. 8, 18

[30] R. Henschel, L. Leal-Taixé, B. Rosenhahn, and K. Schindler. Tracking with multi-level features. *Submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016. 1, 16

[31] R. Henschel, Y. Zou, and B. Rosenhahn. Multiple people tracking using body and joint detections. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019. 1, 8, 18

[32] H. Jiang, S. Fels, and J. Little. A linear programming approach for multiple object tracking. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2007. 2

[33] R. Kasturi, D. Goldgof, P. Soundararajan, V. Manohar, J. Garofolo, M. Boonstra, V. Korzhova, and J. Zhang. Framework for performance evaluation for face, text and vehicle detection and tracking in video: data, metrics, and protocol. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 31(2):319–336, 2009. 6

[34] M. Keuper, S. Tang, B. Andres, T. Brox, and B. Schiele. Motion segmentation & multiple object tracking by correlation co-clustering. *PAMI*, pages 1–1, 2018. 1, 8, 17, 18

[35] M. Keuper, S. Tang, B. Andres, T. Brox, and B. Schiele. Motion segmentation and multiple object tracking by correlation co-clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019. 16, 17

[36] C. Kim, F. Li, A. Ciptadi, and J. Rehg. Multiple hypothesis tracking revisited: Blending in modern appearance model. *ICCV*, 2015. 18

[37] C. Kim, F. Li, and J. M. Rehg. Multi-object tracking with neural gating using bilinear lstm. In *The European Conference on Computer Vision (ECCV)*, September 2018. 6

[38] S. Kim, S. Kwak, J. Feyereisl, and B. Han. Online multi-target tracking by large margin structured learning. pages 98–111, 2013. 2

[39] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel. Neural relational inference for interacting systems. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2688–2697, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR. 4

[40] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 2

[41] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. *ANIPS*, 2012. 2

[42] L. Leal-Taixé, C. Canton-Ferrer, and K. Schindler. Learning by tracking: siamese cnn for robust target association. *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR). DeepVision: Deep Learning for Computer Vision.*, 2016. 2, 4

[43] L. Leal-Taixé, M. Fenzi, A. Kuznetsova, B. Rosenhahn, and S. Savarese. Learning an image-based motion context for multiple people tracking. 4

[44] L. Leal-Taixé, M. Fenzi, A. Kuznetsova, B. Rosenhahn, and S. Savarese. Baseline used in: Learning and image-based motion context for multiple people tracking. 2014. 1

[45] L. Leal-Taixé, M. Fenzi, A. Kuznetsova, B. Rosenhahn, and S. Savarese. Learning an image-based motion context for multiple people tracking. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 2

[46] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler. Motchallenge 2015: Towards a benchmark for multi-target tracking. *arXiv:1504.01942*, 2015. 6, 8, 18

[47] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler. Motchallenge 2015: Towards a benchmark for multi-target tracking. *arXiv:1504.01942*, 2015. 17

[48] L. Leal-Taixé, G. Pons-Moll, and B. Rosenhahn. Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker. *IEEE International Conference on Computer Vision (ICCV) Workshops. 1st Workshop on Modeling, Simulation and Visual Analysis of Large Crowds*, 2011. 1, 2

[49] L. Leal-Taixé, G. Pons-Moll, and B. Rosenhahn. Branch-and-price global optimization for multi-view multi-target tracking. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 2

[50] W. Li, R. Zhao, T. Xiao, and X. Wang. Deepreid: Deep filter pairing neural network for person re-identification. In *CVPR*, 2014. 6

[51] Z. Li, Q. Chen, and V. Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 539–548. Curran Associates, Inc., 2018. 2

[52] C. Ma, C. Yang, F. Yang, Y. Zhuang, Z. Zhang, H. Jia, and X. Xie. Trajectory factory: Tracklet cleaving and reconnection by deep siamese bi-gru for multiple object tracking. *ICME*, abs/1804.04555, 2018. 8

[53] L. Ma, S. Tang, M. Blakc, and L. van Gool. Customized multi-person tracker. 2019. 6, 8, 17, 18

[54] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler. Mot16: A benchmark for multi-object tracking. *arXiv:1603.00831*, 2016. 6, 8, 18

[55] A. Milan, L. Leal-Taixé, K. Schindler, and I. Reid. Joint tracking and segmentation of multiple targets. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 2

[56] M. Narasimhan, S. Lazebnik, and A. G. Schwing. Out of the box: Reasoning with graph convolution nets for factual visual question answering. *CoRR*, abs/1811.00538, 2018. 2

[57] P. Ochs, J. Malik, and T. Brox. Segmentation of moving objects by long term video analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(6):1187–1200, June 2014. 18

[58] S. Pellegrini, A. Ess, K. Schindler, and L. van Gool. You'll never walk alone: modeling social behavior for multi-target tracking. *IEEE International Conference on Computer Vision (ICCV)*, pages 261–268, 2009. 2

[59] H. Pirsiavash, D. Ramanan, and C. Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1201–1208, 2011. 2

[60] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 91–99, Cambridge, MA, USA, 2015. MIT Press. 1, 16

[61] S. Ren, R. G. K. He, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Neural Information Processing Systems (NIPS)*, 2015. 2

[62] E. Ristani, F. Solera, R. S. Zou, R. Cucchiara, and C. Tomasi. Performance measures and a data set for multi-target, multi-camera tracking. *ECCV Workshops*, 2016. 6

[63] E. Ristani and C. Tommasi. Features for multi-target multi-camera tracking and re-identification. *CVPR*, 2018. 2

[64] A. Sadeghian, A. Alahi, and S. Savarese. Tracking the untrackable: Learning to track multiple cues with long-term dependencies. *ICCV*, Oct 2017. 2

[65] A. Sadeghian, A. Alahi, and S. Savarese. Tracking the untrackable: Learning to track multiple cues with long-term dependencies. *ICCV*, abs/1701.01909, 2017. 8

[66] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *Trans. Neur. Netw.*, 20(1):61–80, Jan. 2009. 2

[67] S. Schulter, P. Vernaza, W. Choi, and M. Chandraker. Deep network flow for multi-object tracking. *CVPR*, 2017. 1, 2, 4, 8, 18

[68] H. Sheng, J. Chen, Y. Zhang, W. Ke, Z. Xiong, and J. Yu. Iterative multiple hypothesis tracking with tracklet-level association. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 1–1, 2018. 18

[69] H. Sheng, Y. Zhang, J. Chen, Z. Xiong, and J. Zhang. Heterogeneous association graph fusion for target association in multiple object tracking. *IEEE Transactions on Circuits and Systems for Video Technology*, 2018. 8, 18

[70] J. Son, M. Baek, M. Cho, and B. Han. Multi-object tracking with quadruplet convolutional neural networks. July 2017. 1, 2, 18

[71] S. Tang, B. Andres, M. Andriluka, and B. Schiele. Subgraph decomposition for multi-target tracking. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5033–5041. IEEE, June 2015. 16

[72] S. Tang, B. Andres, M. Andriluka, and B. Schiele. Multi-person tracking by multicuts and deep matching. In *ECCV Workshop on Benchmarking Mutliple Object Tracking*, 2016. 16

[73] S. Tang, M. Andriluka, B. Andres, and B. Schiele. Multiple people tracking by lifted multicut and person re-identification. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3701–3710, Washington, DC, USA, July 2017. IEEE Computer Society. 1, 6, 16

[74] S. Tang, M. Andriluka, B. Andres, and B. Schiele. Multiple people tracking by lifted multicut and person re-identification. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3701–3710, July 2017. 8, 17, 18

[75] S. Tang, M. Andriluka, and B. Schiele. Multi people tracking with lifted multicut and person re-identification. *CVPR*, 2017. 1, 2

[76] S. Wang and C. Fowlkes. Learning optimal parameters for multi-target tracking. *BMVC*, 2015. 2

[77] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. Deepflow: Large displacement optical flow with deep matching. pages 1385–1392, Dec 2013. 18

[78] N. Wojke and D. Paulus. Global data association for the probability hypothesis density filter using network flows. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 567–572, May 2016. 17, 18

[79] Z. Wu, T. Kunz, and M. Betke. Efficient track linking methods for track graphs using network-flow and set-cover techniques. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1185–1192, 2011. 2

[80] J. Xu, Y. Cao, Z. Zhang, and H. Hu. Spatial-temporal relation networks for multi-object tracking. *ICCV*, abs/1904.11489, 2019. 1

[81] J. Xu, Y. Cao, Z. Zhang, and H. Hu. Spatial-temporal relation networks for multi-object tracking. *ICCV*, abs/1904.11489, 2019. 8

[82] F. Yang, W. Choi, and Y. Lin. Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifiers. *CVPR*, 2016. 2

[83] Y.-C. Yoon, A. Boragule, K. Yoon, and M. Jeon. Online multi-object tracking with historical appearance matching and scene adaptive detection filtering. *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6, 2018. 16

[84] Q. Yu, G. Medioni, and I. Cohen. Multiple target tracking using spatio-temporal markov chain monte carlo data association. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2007. 1

[85] A. Zamir, A. Dehghan, and M. Shah. Gmcp-tracker: Global multi-object tracking using generalized minimum clique graphs. *ECCV*, 2012. 2

[86] L. Zhang, Y. Li, and R. Nevatia. Global data association for multi-object tracking using network flows. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2008. 1, 2

[87] L. Zhang, Y. Li, and R. Nevatia. Global data association for multi-object tracking using network flows. *CVPR*, 2008. 1, 2, 3, 12

[88] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian. Scalable person re-identification: A benchmark. In *Proceedings of the IEEE International Conference on Computer Vision*, 2015. 6

[89] J. Zhu, H. Yang, N. Liu, M. Kim, W. Zhang, and M.-H. Yang. Online multi-object tracking with dual matching attention networks. September 2018. 1, 2

Supplementary Material

A. Network Flow Formulation

In this section, we detail how the network flow-based formulation we present in the main paper is related to the original one proposed in [87]. See Figure 4 for an overview.

A.1. Active and Inactive Detections

Let $G = (V, E)$ be a graph representing a multiple object tracking (MOT) problem. In our method's graph formulation, we use nodes and edges to represent, respectively, detections and possible links forming trajectories among them. Moreover, we assign a binary variable $y_{(i,j)}$ to every edge $(i, j) \in E$ to represent whether the link between detections i and j is active or not. Nodes, i.e., detections, do not have any variable assigned to them, and we assume that all detections in the graph are correct. That is, we assume that there are no false positives in the graph[4].

In the general min-cost flow formulation, instead, detections are also represented with edges and they are assigned, in turn, another binary variable that indicates whether the detection is a true or false positive. Formally, for the ith input detection this binary variable is denoted as y_i, and its value is one if the detection is a true positive, i.e., it is active, and zero otherwise, i.e., it is inactive.

By allowing nodes to be inactive, the flow conservation constraints 1 and 2 we described in the main paper:

$$\sum_{(j,i)\in E \text{ s.t. } t_i>t_j} y_{(j,i)} \leq 1 \tag{1}$$

$$\sum_{(i,k)\in E \text{ s.t. } t_i<t_k} y_{(i,k)} \leq 1 \tag{2}$$

are no longer sufficient. Instead, these constraints need to capture that, if a detection is inactive, both its incoming and outgoing flows need to be zero. This is achieved by replacing the right-hand-side of these inequalities with the binary variable y_i:

$$\sum_{(j,i)\in E \text{ s.t. } t_i>t_j} y_{(j,i)} \leq y_i \tag{11}$$

$$\sum_{(i,k)\in E \text{ s.t. } t_i<t_k} y_{(i,k)} \leq y_i \tag{12}$$

Observe that whenever $y_i = 0$, all edges entering and leaving detection i need to be inactive. In contrast, when $y_i = 1$, i.e., the detection is active, these constraints are equivalent to the ones we use.

[4]We can make this assumption because we can easily filter false positives from our graphs during pre-processing and post-processing (see Appendix C).

A.2. Source and Sink Nodes

In the classical min-cost flow formulation of MOT [87], there are two special nodes: *source* and *sink*. Every detection i is connected to both of them, and the resulting edge receives a binary variable denoted as $y_{en,i}$ and $y_{ext,i}$, respectively. These are used to indicate whether a trajectory starts or ends at i. Observe that $y_{en,i} = 1$ if, and only if, there is no detection j in a past frame such that $y_{i,j} = 1$, and analogously for $y_{ext,i} = 1$. Hence, these variables can be used to transform inequalities 11 and 12 into equalities as:

$$y_{en,i} + \sum_{(j,i)\in E \text{ s.t. } t_i>t_j} y_{(j,i)} = y_i \tag{13}$$

$$y_{ext,i} + \sum_{(i,k)\in E \text{ s.t. } t_i<t_k} y_{(i,k)} = y_i \tag{14}$$

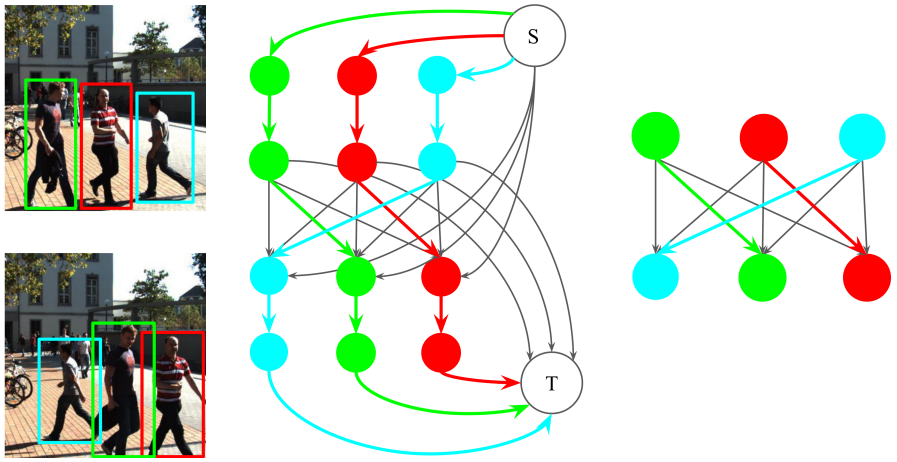
which yield the flow conservation constraints introduced in [87]. In the original min-cost flow formulation, these edges are assigned a handcrafted cost indicating the *price* of starting or ending a trajectory. If this cost is set to zero, one can think about $y_{en,i}$ and $y_{ext,i}$ as *slack* variables.

A.3. Overview of our Simplification

To summarize, in our method we simplify the min-cost flow formulation by eliminating two elements of the classical one: detection edges and sink and source nodes. The first choice allows us to decouple the data association problem from the identification of incorrect detections. Since the latter task can be easily tackled in our pre-processing and post-processing routines (see Appendix C), we can simplify our graph formulation and allow our network to focus on the task of edge classification. As for not using sink and source nodes, in our method there is no need for such *special* variables. Instead, the start (resp. end) of a trajectory is naturally indicated by the absence of active incoming (resp. outgoing) edges to a node. Overall, we simplify the min-cost flow MOT formulation and reduce it to its most essential component: association edges. As a result, we obtain a setting that is suited for our message passing network to operate and effectively learn our task at hand.

B. Rounding Solutions

As explained in the main paper (Section 4.4), a forward pass through our model yields a fractional solution to the original flow problem with values between 0 and 1. Thanks to our time-aware message passing network, binarizing this solution directly by setting a threshold at 0.5 will yield a solution that satisfies close to 99% of the flow conservation constraints on average over test sequences (see Section 5.2 in the main paper). In order to guarantee that all of them



(a) Tracking Output (b) Classical Flow Formulation (c) Simplified Flow Formulation

Figure 4: Overview of the simplification of the classical min-cost flow formulation of MOT used in our method. 4a shows two frames with different trajectories indicated by different bounding box color. 4b shows the classical flow-based view of the scenario: active detections are displayed with *detection* edges, and start (resp. ends) of trajectories are indicated with connections to the source (S) (resp. source (T)) node. 4c shows the formulation used in our approach: no sink nor source nodes are used, and active detections are represented with a single node.

are satisfied, we propose two simple schemes, and describe them in this section. See Figure 5 for a summary of our procedure.

B.1. Greedy Rounding

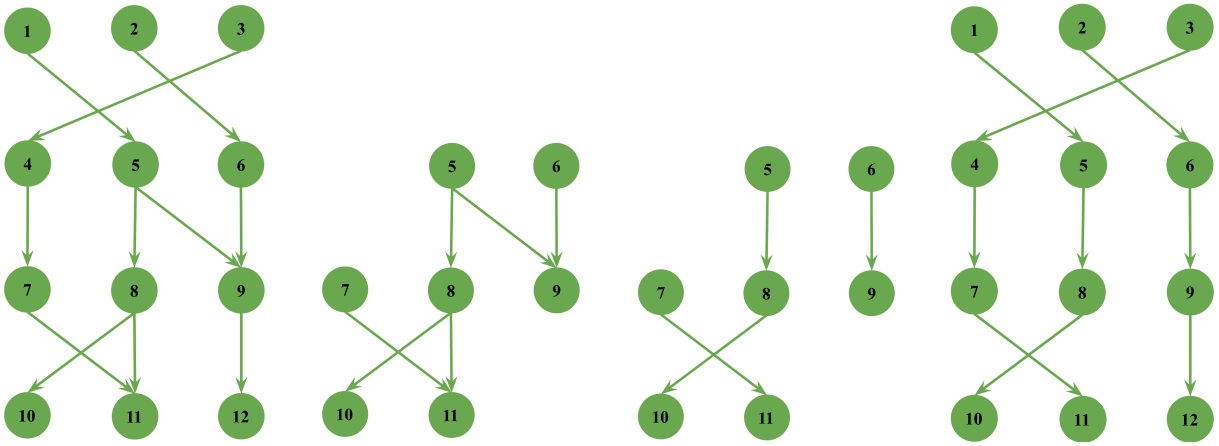
In our setting, having a violated incoming (resp. outgoing) flow conservation constraint means that, for some node, there is more than one incoming (resp. outgoing) edge classified as active. Hence, a simple way to obtain a binary solution satisfying all constraints is to only set as active the incoming (resp. outgoing) edge with the maximum classification score, for every node.

Let $G = (V, E)$ a MOT graph. Observe that, by following this simple policy, we are guaranteed to obtain a binary feasible solution after $o(\Delta(G)|V|)$ steps, where $\Delta(G)$ indicates the maximum degree of any vertex in G. Indeed, observe that we have a total of $2|V|$ constraints, since for each node, there are two flow conservation inequalities. Evaluating each of these requires computing a sum of $o(\Delta(G))$ terms, and picking the edge with maximum score among all neighbors in past / future frames has, again, complexity $o(\Delta(G))$. Further observe that, by picking the edge with the highest classification score no new constraints can be violated. Indeed, setting all non-maximum incoming (resp. outgoing) edges in a node to zero will make all remaining left hand sides in inequalities 1 and 2 for other nodes become smaller or equal. Hence, it is clear that, at most, $2|V|$ iterations with $o(\Delta(G))$ operations each will be necessary, which yields a total complexity of $o(\Delta(G)|V|)$. See Algo-

Algorithm 1: Greedy Rounding

Input : Graph $G = (V, E)$
Fractional solution $\hat{y}$
Result: Binary feasible solution y

```
// Threshold initial solution
foreach (i, j) ∈ E do
    if ŷ_(i,j) ≥ 0.5 then
        y_(i,j) ← 1
    else
        y_(i,j) ← 0
// Iterate over constraints
foreach i ∈ V do
    // Set as inactive all edges but
       the one with max. score
    if Constraint 1 is violated then
        j* ← argmax_{j∈N_i^past} ŷ_(i,j)
        foreach j ∈ N_i^past \ {j*} do
            y_(i,j) ← 0
    if Constraint 2 is violated then
        j* ← argmax_{j∈N_i^fut} ŷ_(i,j)
        foreach j ∈ N_i^fut \ {j*} do
            y_(i,j) ← 0
return y
```



(a) Output Graph Binarized by Thresholding

(b) Subgraph of Violated Constraints

(c) Subgraph of Violated Constraints with Exact / Greedy Rounding

(d) Final Graph

Figure 5: Toy example showing how we round our method's output solutions. 5a shows the output of binarizing our method's output by setting a threshold on edge's classification score at 0.5. Edges' arrows indicate time direction. In the example, we observe 4 violated constraints: outgoing flow is greater than 1 for both nodes 5 and 8, and incoming flow is greater than 1 for nodes 9 and 11. 5b shows the subgraph corresponding to the edges involved in these violated constraints, which requires rounding by either our exact or greedy scheme. 5c shows indicates the result of rounding solutions in the subgraph. Hence, it no longer has constraints violations. 5d shows the final graph, in which both the solutions obtained by thresholding in 5a, and the rounding solutions of the subgraph in 5d are combined to yield the final trajectories.

rithm 1 for a summary of this procedure.

B.2. Exact Rounding

As we show in Table 4, the greedy rounding scheme we just introduced works very well in practice. This is due to the fact that our method's output solutions already satisfy almost all constraints and hence, there is little margin for our rounding scheme to affect performance. However, in general, greedy rounding is not guaranteed to be optimal. We now explain how *exact* rounding can be performed via linear programming.

Let $\hat{y} \in [0,1]^{|E|\times 1}$ denote the fractional output of our network at every edge. Also let $A \in \{0,1\}^{2|V|\times|E|}$ be the matrix resulting from expressing constraints 1 and 2 for all nodes in matrix notation, and $\mathbb{1}_{2|V|}$ a $2|V|-$dimensional column vector of ones corresponding to the constraint's right hand side. The exact rounding problem consists in obtaining the binary solution $y_{int} \in \{0,1\}^{|E|\times 1}$ satisfying constraints 1 and 2 for all entries that is closest (w.r.t the squared euclidean norm) to our networks' output. Hence, in matrix notation exact rounding can be formulated as:

$$\begin{array}{ll} \min_{y_{int}} & \|y_{int} - \hat{y}\|_2^2 \\ \text{subject to} & A y_{int} \leq \mathbb{1}_{2|V|} \\ & y_{int} \in \{0,1\}^{|E|\times 1} \end{array}$$

However, the quadratic cost can be equivalently written as a linear function. Indeed:

$$\begin{aligned} \min_{y_{int}} \|y_{int} - \hat{y}\|_2^2 &= \min_{y_{int}} (y_{int} - \hat{y})^T (y_{int} - \hat{y}) \\ &= \min_{y_{int}} y_{int}^T y_{int} - 2y_{int}^T \hat{y} + \hat{y}^T \hat{y} \\ &= \min_{y_{int}} y_{int}^T y_{int} - 2y_{int}^t \hat{y} \\ &= \min_{y_{int}} y_{int}^T \mathbb{1}_{|E|} - 2y_{int}^T \hat{y} \\ &= \min_{y_{int}} y_{int}^T (\mathbb{1}_{|E|} - 2\hat{y}) \end{aligned}$$

Where $\mathbb{1}_{|E|}$ is an $|E|-$dimensional column vector of ones. Observe that $y_{int}^T y_{int} = y_{int}^T \mathbb{1}_{|E|}$ holds due to the fact that y_{int} is a vector of ones and zeros.

Moreover, matrix A, is *totally unimodular* [23, 3], which implies that the integrality constraints can be relaxed to box constraints $y_{int} \in [0,1]^{|E|\times 1}$. Therefore, we can relax our original quadratic integer program to a linear program, and solve it in polynomial time while still being guaranteed integer solutions.

In general, this optimization problem shows a straightforward connection between our setting and the general min-cost flow problem. When naively rounding our solutions with linear programming, we could view our model's output as edge costs in a min-cost-flow problem instance. The key difference is that, in practice, we do not need to

Rounding	MOTA ↑	IDF1 ↑	MT ↑	ML ↓	FP ↓	FN ↓	ID Sw. ↓	Hz. ↓
Greedy	64.0	69.8	638	362	5703	115089	640	6.5
Exact	64.0	70.0	648	362	6169	114509	602	6.5

Table 4: We compare the effect of having an approximate *greedy* rounding scheme, with respect to performing *exact* rounding via linear programming. Both methods show almost the same performance and no significant difference in speed (Hz, in frames per second).

solve the entire problem. Since our model's output is almost feasible, when rounding, we can obtain binary solutions for almost all edges by directly thresholding their classification scores. With the remaining edges in which flow conservation constraints are violated, we consider their corresponding subgraph, which typically consists of less than 5% of edges in the graph, and either solve the linear program we have described or use the greedy procedure explained in Subsection B.1.

B.3. Performance Comparison

In Table 4, we compare the runtime and performance of both rounding schemes with our model's final configuration. Both *exact* and *greedy* rounding show almost equal performance, with greedy rounding having slightly lower IDF1 (0.2 percentage points), equal MOTA and equal speed. Overall this shows, that given the high constraint satisfaction of our model's solutions, the method for rounding has little effect on the tracking results. This is to be expected: since there are few edges in which the rounding procedure needs to be applied, there is little room to affect overall results. Instead, the key element driving performance in our model is our message passing network formulation.

C. Further Implementation Details

In this section, we extend the information provided in the main article about the implementation of our method.

C.1. Detailed Architecture

In Table 5, we specify the configuration of each of the network's components. Observe that our model is composed of a total of 6 networks. The first two, $\mathcal{N}_v^{enc}$ and $\mathcal{N}_e^{enc}$, are used for feature encoding of nodes and edges, respectively (see section 4.3 in the main paper). For neural message passing, we use one network to update edge embeddings, $\mathcal{N}_e$, and three networks to update node embeddings $\mathcal{N}_v^{past}$, $\mathcal{N}_v^{fut}$ and $\mathcal{N}_v$ (see sections 4.1 and 4.2 in the main paper). Lastly, to classify edges, we use another network, $\mathcal{N}_e^{class}$ (see section 4.4 in the main paper).

	Layer Num.	Type	Output Size
Feature Encoders		Nodes ($\mathcal{N}_v^{enc}$)	
	0	Input	$3 \times 128 \times 64$
	1	conv 7×7	$64 \times 64 \times 32$
	2	max pool 3×3	$64 \times 32 \times 16$
	3	conv1	$256 \times 32 \times 16$
	4	conv2	$512 \times 16 \times 8$
	5	conv3	$1024 \times 8 \times 4$
	6	conv4	$2048 \times 8 \times 4$
	7	GAP	2048
	8	FC + ReLU	512
	9	FC + ReLU	128
	10	FC + ReLU	32
		Edges ($\mathcal{N}_e^{enc}$)	
	0	Input	6
	1	FC + ReLU	18
	2	FC + ReLU	18
	3	FC + ReLU	16
Message Passing Network		Past Update ($\mathcal{N}_v^{past}$)	
	0	Input	80
	1	FC + ReLU	56
	2	FC + ReLU	32
		Future Update ($\mathcal{N}_v^{fut}$)	
	0	Input	80
	1	FC + ReLU	56
	2	FC + ReLU	32
		Node Update ($\mathcal{N}_v$)	
	0	Input	64
	1	FC + ReLU	32
		Edge Update ($\mathcal{N}_e$)	
	0	Input	160
	1	FC + ReLU	80
	2	FC + ReLU	16
Classifier		Edges ($\mathcal{N}_e^{class}$)	
	0	Input	16
	1	FC + ReLU	8
	2	FC + Sigmoid	1

Table 5: For each network component, we specify its number of layers and input / output dimensions. FC denotes a *fully connected* layer and GAP, *Global Average Pooling*. . For the Node Encoder, $\mathcal{N}_v^{enc}$, all layers up to position 7 correspond to those of a ResNet50[28]. Hence, 'layers' 3 to 6 are actually sequences of residual blocks. The only modification we have applied is using stride 1 in all convolutional layers of conv4. Hence, there is no spatial size reduction from conv3 to conv4.

C.2. Batch Processing

As explained in the main paper, we process videos by sequentially feeding overlapping batches of 15 frames to

our model. In the MOTChallenge, different sequences show great variability regarding (i) number of frames per second at which videos are recorded (ii) presence of camera movement and (iii) number of detections per frame. To account for (i) and (i), we sample a fixed and number of frames per second for static and dynamic sequences, which we set to 6 and 9, respectively. To tackle (iii), we restrict the connectivity of graphs by connecting two nodes only if both are among the top-50 reciprocal nearest neighbors according to the ResNet features. This ensures that our model scales to crowded sequences with no significant overhead, and that the topology of our graphs is comparable among different videos.

C.3. Cross-Validation Splits

We conduct all of our experiments with 3-fold cross-validation on the MOT17 benchmark training data. To do so, we split the sequences into three subsets. For each experiment configuration we train a total of 3 networks: one for each possible validation set. Since our splits cover all training sequences of MOT17, we obtain metrics over the whole dataset which allow us to choose the best network configuration and set of hyperparameters.

In Table 6, we report the validation sequences corresponding to each split. For each of the splits, the sequences not contained in its validation set are used for training, together with those of the MOT15 dataset.

When deciding which sequences to include in each split, we made sure that each subset contains both moving and static camera sequences. Furthermore, we balance the number of tracks and sequence length in seconds (recall that fps is normalized during processing) in each split, in order to ensure that all validation settings are comparable.

Name	Mov.	Length (s)	Length (f)	Tracks	Boxes
Split 1					
MOT17-02	No	20	600	62	18581
MOT17-10	Yes	22	654	57	12839
MOT17-13	Yes	30	750	110	11642
Overall	–	72	2004	229	43062
Split 2					
MOT17-04	No	35	1050	83	47557
MOT17-11	Yes	30	900	75	9436
Overall	–	65	1950	158	56993
Split 3					
MOT17-05	Yes	60	837	133	6917
MOT17-09	No	18	525	26	5325
Overall	–	78	1362	159	12242
Total					
	–	205	5316	546	112297

Table 6: We report the sequences used in each cross-validation split in order to evaluate our model. *Mov* refers to whether there is camera movement in the scene and, for length 's' denotes seconds and 'f', frames. Note that in the MOT17 benchmark, each sequence is given with three sets of detections (DPM, FRCNN and SDP). Since the ground truth does not change among them, here we report the features of each sequence, and do not take into account the set of detections. However, when testing our models, we make use of all sets of detections.

C.4. Preprocessing and Postprocessing

As we explain in the next section we use [9] to preprocess public detections. As an alternative, for the results in Section D, we follow a similar detection preprocessing scheme to the one applied by other methods [14, 83, 35]. We use both the bounding box regressor and classifier heads of a Faster-RCNN[60] trained on the MOT17 Detection challenge. We filter out all bounding boxes whose confidence score is smaller than 0.5, and correct the remaining with the bounding box regressor. After that, we apply standard Non-Maxima-Supression to the resulting boxes, by using each box' confidence score, and setting an IoU threshold of 0.85. For post-processing, if using [9] we fill gaps in our trajectories by matching our output trajectories to the ones in [9], and then using the latter to fill the detections in missing frames. For the remaining missing gaps in our trajectories, we use bilinear interpolation. Finally, we drop all trajectories that consist of a single detection. This allows our model to identify false positives as *isolated* nodes in the graph (i.e. nodes with neither incoming nor outgoing active edges).

C.5. Baseline

As explained in the main paper, we use [9] as a baseline. More specifically, we preprocess all sequences by first running [9] on public detections. After that, we discard the pedestrian ID assigned by [9], and simply treat the resulting boxes as raw detections for our neural solver. [9] uses the regression head of a Faster-R-CNN [60] in order to predict the next locations of objects in neighboring frames

Other graph approaches resort to low-level image features and work with raw (i.e. non-maxima suppressed) detections to approach this challenge [30, 72, 71, 73, 35]. Observe that using raw detections has indeed, more potential than just adding neighboring detections with [9], as it yields a greatly increased number of object hypothesis. Hence, it allows tracking methods to have the capacity to track more objects. However, [9] reduces computational times significantly, and provides more precise boxes, which improves the efficiency of our method.We perform a detailed comparison with graph-based methods in the next section.

D. Additional Comparison with Graph Methods

We provide an extended comparison of our method[5] with top-performing offline graph-based methods. The results are summarized in Table 7. For each method, we highlight the additional features and sources of information that it has access to. Additionally, we provide the results obtained by our method when we do not use our baseline [9] for preprocessing detections, and we denote it with *Ours**. We show that, even in that case, our method still surpasses previous works by a significant margin even though it has access to significantly less information. Hence, these results further confirm the superiority of our approach.

Even without [9], in the MOT15[47] dataset we observe an improvement of 19.8 points in MOTA and 15.6 points in IDF1 with respect to [78], which uses the same underlying Min-Cost Flow graph formulation, but a simpler learning scheme. Moreover, in all three datasets, our method consistently improves significantly upon multi-cut based methods [34, 53, 35, 74], which use a more involved graph formulation, have access to a significantly larger number of boxes due to not using Non-Maximum Suppression, and either employ low-level image features or use a hierarchical scheme. Thus, we clearly demonstrate that our method shows very strong performance and surpasses previous work, even when it cannot leverage low-level image information via [9]. Furthermore, when our method is given access to additional features as other methods, it shows its full potential and outperforms all previous works by an even larger margin.

[5] We made a slight change in the configuration of our method for these results. Since we do not have access to [9] and, hence, we have to rely heavily on linear interpolation for postprocessing (see Appendix C.4), we augment the frame sampling rate at which we process sequences, and also the size of graphs we process proportionally, in order to cover time intervals of the same size of those of our main configuration. Specifically, we increase the sampling rate of frames for static sequences from 6 to 9, and from 9 to 15 for those with a moving camera. As for the number of frames corresponding to each processed graph, we increase it from 15 to 25.

Method	MOTA ↑	IDF1 ↑	MT ↑	ML ↓	ID Sw. ↓	Hz ↑	Additional Features
2D MOT 2015 [46]							
Ours	**51.5**	**58.6**	**31.2**	**25.9**	**375**	6.5	Box regression [9]
Ours*	46.6	53.8	25.2	29.0	381	6.5	———
JointMC [34]	35.6	45.1	23.2	39.3	457	0.6	Point trajectories [57], opt. flow
QuadMOT [70]	33.8	40.4	12.9	36.9	703	3.7	Learnable box regression
MHT_DAM [36]	32.4	45.3	16.0	43.8	435	0.7	———
MCF_PHD [78]	29.9	38.2	11.9	44.0	656	12.2	———
DeepFlow [67]	26.8	–	–	–	–	–	ALFD motion features [16]
MOT16 [54]							
Ours	**58.6**	**61.7**	**27.3**	**34.0**	**354**	6.5	Box regression [9]
Ours*	54.3	56.8	23.5	36.0	440	6.5	———
NOTA [12]	49.8	55.3	17.9	37.7	614	–	Not public
HCC [53]	49.3	50.7	17.8	39.9	391	0.8	Tracklets, Deep Matching [77]
LMP [74]	48.8	51.3	18.2	40.1	481	0.5	Body part detections, Deep Matching [77], no-NMS
TLMHT [68]	48.7	55.3	15.7	44.5	413	4.8	Tracklets
FWT [29]	47.8	44.3	19.1	38.2	852	0.6	Head detections, Deep Matching [77]
MOT17 [54]							
Ours	**58.8**	**61.7**	**28.8**	**33.5**	**1185**	6.5	Box regression [9]
Ours*	56.0	58.4	26.2	34.7	1451	6.5	———
JBNOT [31]	52.6	50.8	19.7	35.8	3050	5.4	Joint detections, Deep Matching [77]
eHAF[69]	51.8	54.7	23.4	37.9	1834	0.7	Superpixels, Segmentation, Foreground Extraction
NOTA [12]	51.3	54.7	17.1	35.4	2285	–	Not public
FWT [29]	51.3	47.6	21.4	35.2	2648	0.2	Head detections, Deep Matching [77]
jCC [34]	51.2	54.5	20.9	37.0	1802	1.8	Point trajectories [57], opt. flow

Table 7: We compare both our final method (Ours) and a variant of our method in which we do not exploit our baseline [9] (Ours*) to other top-performing graph-based offline methods in the MOTChallenge benchmark. Under *Additional Features*, we highlight which information sources or features each method used apart from the given public detections. We denote with a horizontal line the absence of such features. For [12], we cannot provide this information, since the article is not freely available. For [67], we report the only metric that the authors reported in their article. We still include it the table due to the fact that it also follows the min-cost flow MOT formulation, and is similar in spirit to our work (see Related Work in the main article).