Learning a Neural Solver for Multiple Object Tracking
– Supplementary Material –

Guillem Brasó* Laura Leal-Taixé

Technical University of Munich

Abstract

As announced in the main paper, in this document we provide: (i) a more exhaustive explanation of the network-flow formulation on which our method is based (Section 1), (ii) a description of our rounding scheme (Section 2), (iii) additional implementation details (Section 3), (iv) a comparison of our method with previous graph-based MOT works (Section 4).

1. 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 [31]. See Figure 1 for an overview.

1.1. 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[1].

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.

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

[1] We can make this assumption because we can easily filter false positives from our graphs during pre-processing and post-processing (see Section 3).

By allowing nodes to be inactive, the flow conservation constraints 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:

$$\sum_{(j,i)\in E \text{ s.t. } t_i>t_j} y_{(j,i)} \leq y_i \tag{3}$$

$$\sum_{(i,k)\in E \text{ s.t. } t_i<t_k} y_{(i,k)} \leq y_i \tag{4}$$

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.

1.2. Source and Sink Nodes

In the classical min-cost flow formulation of MOT [31], 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 3 and 4 into equalities as:

$$y_{en,i} + \sum_{(j,i)\in E \text{ s.t. } t_i>t_j} y_{(j,i)} = y_i \tag{5}$$

$$y_{ext,i} + \sum_{(i,k)\in E \text{ s.t. } t_i<t_k} y_{(i,k)} = y_i \tag{6}$$

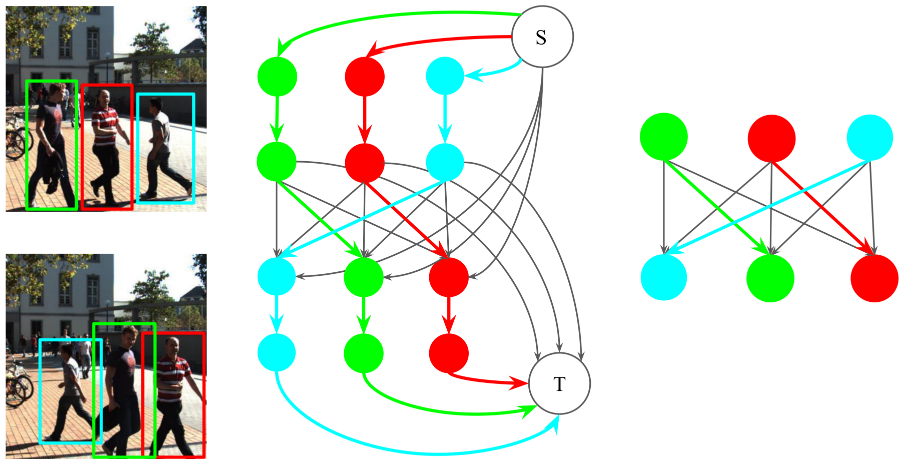

S
T
(a) Tracking Output
(b) Classical Flow Formulation
(c) Simplified Flow Formulation

Figure 1: Overview of the simplification of the classical min-cost flow formulation of MOT used in our method. 1a shows two frames with different trajectories indicated by different bounding box color. 1b 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. 1c shows the formulation used in our approach: no sink nor source nodes are used, and active detections are represented with a single node.

which yield the flow conservation constraints introduced in [31]. 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.

1.3. 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 Section 3), 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.

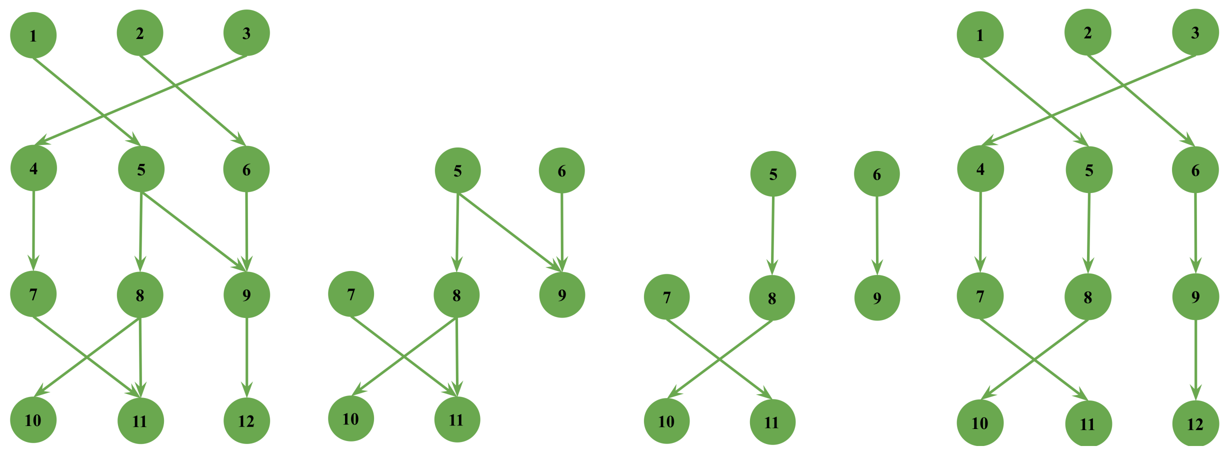
2. Rounding Solutions

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 are satisfied, we propose two simple schemes, and describe them in this section. See Figure 2 for a summary of our procedure.

2.1. 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.



(a) Output Graph Binarized by Thresholding

(b) Subgraph of Violated Constraints

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

(d) Final Graph

Figure 2: Toy example showing how we round our method's output solutions. 2a 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. 2b shows the subgraph corresponding to the edges involved in these violated constraints, which requires rounding by either our exact or greedy scheme. 2c shows indicates the result of rounding solutions in the subgraph. Hence, it no longer has constraints violations. 2d shows the final graph, in which both the solutions obtained by thresholding in 2a, and the rounding solutions of the subgraph in 2d are combined to yield the final trajectories.

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 Algorithm 1 for a summary of this procedure.

2.2. Exact Rounding

As we show in Table 1, 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:

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

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* [6, 1], 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 integer program to a linear program, and solve it in polynomial time while still being guaranteed integer solutions.

Algorithm 1: Greedy Rounding

```
Input : Graph $G = (V, E)$
        Fractional solution $\hat{y}$
Result: Binary feasible solution $y$
1  // Threshold initial solution
2  foreach $(i, j) \in E$ do
3      if $\hat{y}_{(i,j)} > 0.5$ then
4          $y_{(i,j)} \leftarrow 1$
5      else
6          $y_{(i,j)} \leftarrow 0$
7  // Iterate over constraints
8  foreach $i \in V$ do
9      // Set as inactive all edges but
           the one with max. score
10     if Constraint 1 is violated then
11         $j_* \leftarrow argmax_{j \in N_i^{past}} \hat{y}_{(i,j)}$
12         foreach $j \in N_i^{past} \setminus \{j_*\}$ do
13             $y_{(i,j)} = 0$
14     if Constraint 2 is violated then
15         $j_* \leftarrow argmax_{j \in N_i^{fut}} \hat{y}_{(i,j)}$
16         foreach $j \in N_i^{fut} \setminus \{j_*\}$ do
17             $y_{(i,j)} = 0$
18 return $y$
```

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 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 2.1.

2.3. Performance Comparison

In Table 1, 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.

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 1: 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 a slight change in runtime (Hz, in frames per second).

3. Further Implementation Details

In this section, we extend the information provided in the main article about the implementation of our method. As announced, in case of acceptance our source code will be made publicly available.

3.1. Detailed Architecture

In Table 2, 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).

3.2. 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 pretrained CNN 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, which boosts our model's generalization capacity.

	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 2: 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[7]. 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.

3.3. 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 3, 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 scenes 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 3: We report the sequences used in each cross-validation split in order to evaluate our model. *Mov* refers to whether there is camaera 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.

3.4. Pre-processing and Post-processing

As we explain in the next section we use [2] to preprocess public detections. As an alternative, for the results in Section 4, we follow a similar detection preprocessing scheme to the one applied by other methods [4, 30, 12]. We use both the bounding box regressor and classifier heads of a Faster-RCNN[19] 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 [2] we fill gaps in our trajectories by matching our output trajectories to the ones in [2], 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).

3.5. Baseline

As explained in the main paper, we use [2] as a baseline. More specifically, we preprocess all sequences by first running [2] on public detections. After that, we discard the pedestrian ID assigned by [2], and simply treat the resulting boxes as raw detections for our neural solver. [2] uses the regression head of a Faster-R-CNN [19] 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 [9, 25, 24, 26, 12]. Observe that using raw detections has indeed, more potential than just adding neighboring detections with [2], as it yields a greatly increased number of object hypothesis. Hence, it allows tracking methods to have the capacity to track more objects. However, [2] 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.

4. Additional Comparison with Graph Methods

As announced in the main paper, we provide an extended comparison of our method[2] with top-performing offline graph-based methods. The results are summarized in Table 4. 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 [2] 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 confirm the superiority of our approach at using restricted feature information.

Even without [2], in the MOT15[15] dataset we observe an improvement of 19.8 points in MOTA and 15.6 points in IDF1 with respect to [29], 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 correlation clustering and graph multi-cut based methods [11, 16, 12, 27], 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 [2]. 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.

[2]We made a slight change in the configuration of our method for these results. Since we do not have access to [2] and, hence, we have to rely heavily on linear interpolation for postprocessing (see Section 3.4), 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.

References

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

[2] P. Bergmann, T. Meinhardt, and L. Leal-Taixé. Tracking without bells and whistles. *The International Conference on Computer Vision(ICCV)*, abs/1903.05625, 2019. 5, 6, 7

[3] 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. 7

[4] 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. 5

[5] W. Choi. Near-online multi-target tracking with aggregated local flow descriptor. *ICCV*, 2015. 7

[6] 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. 3

[7] 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. 5

[8] 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. 7

[9] R. Henschel, L. Leal-Taixé, B. Rosenhahn, and K. Schindler. Tracking with multi-level features. *Submitted to IEEE Trans-*

Method	MOTA ↑	IDF1 ↑	MT ↑	ML ↓	ID Sw. ↓	Hz ↑	Additional Features
			2D MOT 2015 [14]				
Ours	**51.5**	**58.6**	**31.2**	**25.9**	**375**	6.5	Box regression [2]
Ours*	46.6	53.8	25.2	29.0	381	6.5	———
JointMC [11]	35.6	45.1	23.2	39.3	457	0.6	Point trajectories [18], opt. flow, no-NMS
QuadMOT [23]	33.8	40.4	12.9	36.9	703	3.7	Learnable box regression
MHT_DAM [13]	32.4	45.3	16.0	43.8	435	0.7	———
MCF_PHD [29]	29.9	38.2	11.9	44.0	656	12.2	———
DeepFlow [20]	26.8	–	–	–	–	–	ALFD motion features [5]
			MOT16 [17]				
Ours	**58.6**	**61.7**	**27.3**	**34.0**	**354**	6.5	Box regression [2]
Ours*	54.3	56.8	23.5	36.0	440	6.5	———
NOTA [3]	49.8	55.3	17.9	37.7	614	–	Not public
HCC [16]	49.3	50.7	17.8	39.9	391	0.8	Tracklets, Deep Matching [28]
LMP [27]	48.8	51.3	18.2	40.1	481	0.5	Body part detections, Deep Matching [28], no-NMS
TLMHT [21]	48.7	55.3	15.7	44.5	413	4.8	Tracklets
FWT [8]	47.8	44.3	19.1	38.2	852	0.6	Head detections, Deep Matching [28]
			MOT17 [17]				
Ours	**58.8**	**61.7**	**28.8**	**33.5**	**1185**	6.5	Box regression [2]
Ours*	56.0	58.4	26.2	34.7	1451	6.5	———
JBNOT [10]	52.6	50.8	19.7	35.8	3050	5.4	Joint detections, Deep Matching [28]
eHAF[22]	51.8	54.7	23.4	37.9	1834	0.7	Superpixels, Segmentation, Foreground Extraction
NOTA [3]	51.3	54.7	17.1	35.4	2285	–	Not public
FWT [8]	51.3	47.6	21.4	35.2	2648	0.2	Head detections, Deep Matching [28]
jCC [11]	51.2	54.5	20.9	37.0	1802	1.8	Point trajectories [18], opt. flow

Table 4: We compare both our final method (Ours) and a variant of our method in which we do not exploit our baseline [2] (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 [3], we cannot provide this information, since the article is not freely available. For [20], 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).

actions on Pattern Analysis and Machine Intelligence, 2016. 6

[10] R. Henschel, Y. Zou, and B. Rosenhahn. Multiple people tracking using body and joint detections. In *CVPR Workshops*, 2019. 7

[11] 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. 6, 7

[12] 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. 5, 6

[13] C. Kim, F. Li, A. Ciptadi, and J. Rehg. Multiple hypothesis tracking revisited: Blending in modern appearance model. *ICCV*, 2015. 7

[14] 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. 7

[15] 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

[16] L. Ma, S. Tang, M. Blakc, and L. van Gool. Customized multi-person tracker. 2019. 6, 7

[17] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler. Mot16: A benchmark for multi-object tracking. *arXiv:1603.00831*, 2016. 7

[18] 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. 7

[19] 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. 6

[20] S. Schulter, P. Vernaza, W. Choi, and M. Chandraker. Deep network flow for multi-object tracking. *CVPR*, 2017. 7

[21] 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. 7

[22] 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. 7

[23] J. Son, M. Baek, M. Cho, and B. Han. Multi-object tracking with quadruplet convolutional neural networks. July 2017. 7

[24] 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. 6

[25] 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. 6

[26] 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. 6

[27] 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. 6, 7

[28] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. Deepflow: Large displacement optical flow with deep matching. In *2013 IEEE International Conference on Computer Vision*, pages 1385–1392, Dec 2013. 7

[29] 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. 6, 7

[30] 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. 5

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