Deep Network Flow for Multi-Object Tracking

Samuel Schulter Paul Vernaza Wongun Choi Manmohan Chandraker
NEC Laboratories America, Media Analytics Department
Cupertino, CA, USA
{samuel,pvernaza,wongun,manu}@nec-labs.com

Abstract

Data association problems are an important component of many computer vision applications, with multi-object tracking being one of the most prominent examples. A typical approach to data association involves finding a graph matching or network flow that minimizes a sum of pairwise association costs, which are often either hand-crafted or learned as linear functions of fixed features. In this work, we demonstrate that it is possible to learn features for network-flow-based data association via backpropagation, by expressing the optimum of a smoothed network flow problem as a differentiable function of the pairwise association costs. We apply this approach to multi-object tracking with a network flow formulation. Our experiments demonstrate that we are able to successfully learn all cost functions for the association problem in an end-to-end fashion, which outperform hand-crafted costs in all settings. The integration and combination of various sources of inputs becomes easy and the cost functions can be learned entirely from data, alleviating tedious hand-designing of costs.

1. Introduction

Multi-object tracking (MOT) is the task of predicting the trajectories of all object instances in a video sequence. MOT is challenging due to occlusions, fast moving objects or moving camera platforms, but it is an essential module in many applications like action recognition, surveillance or autonomous driving. The currently predominant approach to MOT is tracking-by-detection [3, 7, 10, 15, 26, 33, 41], where, in a first step, object detectors like [16, 43, 51] provide potential locations of the objects of interest in the form of bounding boxes. Then, the task of multi-object tracking translates into a data association problem where the bounding boxes are assigned to trajectories that describe the path of individual object instances over time.

Bipartite graph matching [25, 35] is often employed in on-line approaches to assign bounding boxes in the current frame to existing trajectories [22, 37, 38, 52]. Off-line methods can be elegantly formulated in a network flow framework to solve the association problem including birth and death of trajectories [27, 29, 54]. Section 2 gives more examples. All these association problems can be solved in a linear programming (LP) framework, where the constraints are given by the problem. The interplay of all variables in the LP, and consequently their costs, determines the success of the tracking approach. Hence, designing good cost functions is crucial. Although cost functions are hand-crafted in most prior work, there exist approaches for learning costs from data. However, they either do not treat the problem as a whole and only optimize parts of the costs [27, 31, 52, 54] or are limited to linear cost functions [49, 50].

We propose a novel formulation that allows for learning *arbitrary parameterized cost functions* for *all variables* of the association problem in an end-to-end fashion, *i.e.*, from input data to the *solution of the LP*. By smoothing the LP, bi-level optimization [6, 13] enables learning of all the parameters of the cost functions such as to minimize a loss that is defined on the solution of the association problem, see Section 3.2. The main benefit of this formulation is its flexibility, general applicability to many problems and the avoidance of tedious hand-crafting of cost functions. Our approach is not limited to log-linear models (*c.f.*, [49]) but can take full advantage of any differentiable parameterized function, *e.g.*, neural networks, to predict costs. Indeed, our formulation can be integrated into any deep learning framework as one particular layer that solves a linear program in the forward pass and back-propagates gradients w.r.t. the costs through its solution (see Figure 2).

While our approach is general and can be used for many association problems, we explore its use for multi-object tracking with a network flow formulation (see Sections 3.1 and 3.4). We empirically demonstrate on public data sets [17, 28, 32] that: (i) Our approach enables end-to-end learning of cost functions for the network flow problem. (ii) Integrating different types of input sources like bounding box information, temporal differences, appearance and motion features becomes easy and all model parameters can be learned jointly. (iii) The end-to-end learned cost func-

tions outperform hand-crafted functions without the need to hand-tune parameters. (iv) We achieve encouraging results with appearance features, which suggests potential benefits from end-to-end integration of deep object detection and tracking, as enabled by our formulation.

2. Related Work

Association problems in MOT: Recent works on multi-object tracking (MOT) mostly follow the tracking-by-detection paradigm [3, 7, 10, 15, 26, 33, 41], where objects are first detected in each frame and then associated over time to form trajectories for each object instance. On-line methods like [8, 11, 15, 39, 41] associate detections of the incoming frame immediately to existing trajectories and are thus appropriate for real-time applications[1]. Trajectories are typically treated as state-space models like Kalman [21] or particle filters [18]. The association to bounding boxes in the current frame is often formulated as bipartite graph matching and solved via the Hungarian algorithm [25, 35]. While on-line methods only have access to the past and current observations, off-line (or batch) approaches [3, 9, 20, 1, 40, 54] also consider future frames or even the whole sequence at once. Although not applicable for real-time applications, the advantage of batch methods is the temporal context allowing for more robust and non-greedy predictions. An elegant solution to assign trajectories to detections is the network flow formulation [54] (see Section 3.1 for details). Both of these association models can be formulated as linear program.

Cost functions: Independent of the type of association model, a proper choice of the cost function is crucial for good tracking performance. Many works rely on carefully designed but hand-crafted functions. For instance, [29, 33, 41] only rely on detection confidences and spatial (*i.e.*, bounding box differences) and temporal distances. Zhang *et al*. [54] and Zamir *et al*. [53] include appearance information via color histograms. Other works explicitly learn affinity metrics, which are then used in their tracking formulation. For instance, Li *et al*. [31] build upon a hierarchical association approach where increasingly longer tracklets are combined into trajectories. Affinities between tracklets are learned via a boosting formulation from various hand-crafted inputs including length of trajectories and color histograms. This approach is extended in [26] by learning affinities on-line for each sequence. Similarly, Bae and Yoon [2] learn affinities on-line with a variant of linear discriminant analysis. Song *et al*. [47] train appearance models on-line for individual trajectories when they are isolated, which can then be used to disambiguate from other trajectories in difficult situations like occlusions or interactions. Leal-Taixé *et al*. [27] train a Siamese neural network to compare the appearance (raw RGB patches) of two detections and combine this with spatial and temporal differences in a boosting framework. These pair-wise costs are used in a network flow formulation similar to [29]. In contrast to our approach, none of these methods consider the actual inference model during the learning phase but rely on surrogate loss functions for parts of the tracking costs.

Integrating inference into learning: Similar to our approach, there have been recent works that also include the full inference model in the training phase. In particular, structured SVMs [48] have recently been used in the tracking context to learn costs for bipartite graph matching in an on-line tracker [23], a divide-and-conquer tracking strategy [46] and a joint graphical model for activity recognition and tracking [12]. In a similar fashion, [49] present a formulation to jointly learn all costs in a network flow graph with a structured SVM, which is the closest work to ours. It shows that properly learning cost functions for a relatively simple model can compete with complex tracking approaches. However, the employed structured SVM limits the cost functions to a linear parameterization. In contrast, our approach relies on bi-level optimization [6, 13] and is more flexible, allowing for non-linear (differentiable) cost functions like neural networks. Bi-level optimization has also been used recently to learn costs of graphical models, *e.g.*, for segmentation [42] or depth map restoration [44, 45].

3. Deep Network Flows for Tracking

We demonstrate our end-to-end formulation for association problems with the example of network flows for multi-object tracking. In particular, we consider a tracking-by-detection framework, where potential detections $\mathbf{d}$ in every frame t of a video sequence are given. Each detection consists of a bounding box $\mathbf{b}(\mathbf{d})$ describing the spatial location, a detection probability $p(\mathbf{d})$ and a frame number $t(\mathbf{d})$. For each detection, the tracking algorithm needs to either associate it with an object trajectory $\mathcal{T}_k$ or reject it. A trajectory is defined as a set of detections belonging to the same object, *i.e.*, $\mathcal{T}_k = \{\mathbf{d}_k^1, \dots, \mathbf{d}_k^{N_k}\}$, where N_k defines the size of the trajectory. Only bounding boxes from different frames can belong to the same trajectory. The number of trajectories $|\mathcal{T}|$ is unknown and needs to be inferred as well.

In this work, we focus on the network flow formulation from Zhang *et al*. [54] to solve the association problem. It is a popular choice [27, 29, 30, 49] that works well in practice and can be solved via linear programming (LP). Note that bipartite graph matching, which is typically used for on-line trackers, can also be formulated as a network flow, making our learning approach equally applicable.

3.1. Network Flow Formulation

We present the formulation of the directed network flow graph with an example illustrated in Figure 1. Each de-

[1] In this context, real-time refers to a causal system.

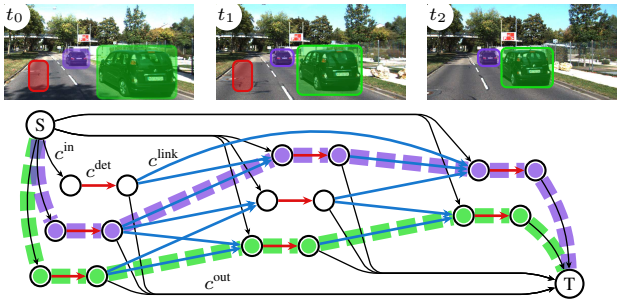

Figure 1: A network flow graph for tracking 3 frames [54]. Each pair of nodes corresponds to a detection. The different solid edges are explained in the text, the thick dashed lines illustrate the solution of the network flow.

tection $\mathbf{d}_i$ is represented with two nodes connected by an edge (red). This edge is assigned the flow variable x_i^{det}. To be able to associate two detections, meaning they belong to the same trajectory $\mathcal{T}$, directed edges (blue) from all $\mathbf{d}_i$ (second node) to all $\mathbf{d}_j$ (first node) are added to the graph if $t(\mathbf{d}_i) < t(\mathbf{d}_j)$ and $|t(\mathbf{d}_i) - t(\mathbf{d}_j)| < \tau_t$. Each of these edges is assigned a flow variable $x_{i,j}^{\text{link}}$. Having edges over multiple frames allows for handling occlusions or missed detections. To reduce the size of the graph, we drop edges between detections that are spatially far apart. This choice relies on the smoothness assumption of objects in videos and does not hurt performance but reduces inference time. In order to handle birth and death of trajectories, two special nodes are added to the graph. A source node (S) is connected with the first node of each detection $\mathbf{d}_i$ with an edge (black) that is assigned the flow variable x_i^{in}. Similarly, the second node of each detection is connected with a sink node (T) and the corresponding edge (black) is assigned the variable x_i^{out}.

Each variable in the graph is associated with a cost. For each of the four variable types we define the corresponding cost, *i.e.*, c^{in}, c^{out}, c^{det} and c^{link}. For ease of explanation later, we differentiate between unary costs $\mathbf{c}^{\text{U}}$ (c^{in}, c^{out} and c^{det}) and pairwise costs c^{P} (c^{link}). Finding the globally optimal minimum cost flow can be formulated as the linear program

$$\begin{aligned} \mathbf{x}^* = \ & \arg\min_{\mathbf{x}} \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \ & \mathbf{A}\mathbf{x} \leq \mathbf{b}, \ \mathbf{C}\mathbf{x} = \mathbf{0}, \end{aligned} \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^M$ and $\mathbf{c} \in \mathbb{R}^M$ are the concatenations of all flow variables and costs, respectively, and M is the problem dimension. Note that we already relaxed the actual integer constraint on $\mathbf{x}$ with box constraints $0 \leq \mathbf{x} \leq 1$, modeled by $\mathbf{A} = [\mathbf{I}, -\mathbf{I}]^\top \in \mathbb{R}^{2M \times M}$ and $\mathbf{b} = [\mathbf{1}, \mathbf{0}]^\top \in \mathbb{R}^{2M}$ in (1). The flow conservation constraints, $x_i^{\text{in}} + \sum_j x_{ji}^{\text{link}} = x_i^{\text{det}}$ and $x_i^{\text{out}} + \sum_j x_{ij}^{\text{link}} = x_i^{\text{det}}$ $\forall i$, are modeled with $\mathbf{C} \in \mathbb{R}^{2K \times M}$, where K is the number of detections. The thick dashed lines in Figure 1 illustrate $\mathbf{x}^*$.

The most crucial part in this formulation is to find proper costs $\mathbf{c}$ that model the interplay between birth, existence, death and association of detections. The final tracking result mainly depends on the choice of $\mathbf{c}$.

3.2. End-to-end Learning of Cost Functions

The main contribution of this paper is a flexible framework to learn functions that predict the costs of *all* variables in the network flow graph. Learning can be done end-to-end, *i.e.*, from the input data all the way to the solution of the network flow problem. To do so, we replace the constant costs $\mathbf{c}$ in Equation (1) with parameterized cost functions $\mathbf{c}(\mathbf{f}, \mathbf{\Theta})$, where $\mathbf{\Theta}$ are the parameters to be learned and $\mathbf{f}$ is the input data. For the task of MOT, the input data typically are bounding boxes, detection scores, images features, or more specialized and effective features like ALFD [10].

Given a set of ground truth network flow solutions $\mathbf{x}^{\text{gt}}$ of a tracking sequence (we show how to define ground truth in Section 3.3) and the corresponding input data $\mathbf{f}$, we want to learn the parameters $\mathbf{\Theta}$ such that the network flow solution minimizes some loss function. This can be formulated as the bi-level optimization problem

$$\begin{aligned} & \arg\min_{\Theta} \mathcal{L}\left(\mathbf{x}^{\text{gt}}, \mathbf{x}^*\right) \\ & \text{s.t.} \ \ \mathbf{x}^* = \arg\min_{\mathbf{x}} \mathbf{c}(\mathbf{f}, \Theta)^\top \mathbf{x} \\ & \qquad \mathbf{A}\mathbf{x} \leq \mathbf{b}, \ \mathbf{C}\mathbf{x} = \mathbf{0}, \end{aligned} \tag{2}$$

which tries to minimize the loss function $\mathcal{L}$ (upper level problem) w.r.t. the solution of another optimization problem (lower level problem), which is the network flow in our case, *i.e.*, the inference of the tracker. To compute gradients of the loss function w.r.t. the parameters $\mathbf{\Theta}$ we require a smooth lower level problem. The box constraints, however, render it non-smooth.

3.2.1 Smoothing the lower level problem

The box constraints in (1) and (2) can be approximated via log-barriers [5]. The inference problem then becomes

$$\begin{aligned} \mathbf{x}^* = \ & \arg\min_{\mathbf{x}} t \cdot \mathbf{c}(\mathbf{f}, \Theta)^\top \mathbf{x} - \sum_{i=1}^{2M} \log(b_i - \mathbf{a}_i^\top \mathbf{x}) \\ \text{s.t.} \ & \mathbf{C}\mathbf{x} = \mathbf{0}, \end{aligned} \tag{3}$$

where t is a temperature parameter (defining the accuracy of the approximation) and $\mathbf{a}_i^\top$ are rows of $\mathbf{A}$. Moreover, we can get rid of the linear equality constraints with a change of basis $\mathbf{x} = \mathbf{x}(\mathbf{z}) = \mathbf{x}_0 + \mathbf{B}\mathbf{z}$, where $\mathbf{C}\mathbf{x}_0 = \mathbf{0}$ and $\mathbf{B} = \mathcal{N}(\mathbf{C})$, *i.e.*, the null space of $\mathbf{C}$, making our objective unconstrained in $\mathbf{z}$ ($\mathbf{C}\mathbf{x} = \mathbf{C}\mathbf{x}_0 + \mathbf{C}\mathbf{B}\mathbf{z} = \mathbf{C}\mathbf{x}_0 = \mathbf{0} =$ True $\forall \mathbf{z}$). This results in the following unconstrained and

smooth lower level problem

$$\arg\min_{\mathbf{z}} t \cdot \mathbf{c}(\mathbf{f}, \Theta)^\top \mathbf{x}(\mathbf{z}) + P(\mathbf{x}(\mathbf{z})), \tag{4}$$

where $P(\mathbf{x}) = -\sum_{i=1}^{2M} \log(b_i - \mathbf{a}_i^\top \mathbf{x})$.

3.2.2 Gradients with respect to costs

Given the smoothed lower level problem (4), we can define the final learning objective as

$$\begin{aligned} &\arg\min_{\boldsymbol{\Theta}} \mathcal{L}\left(\mathbf{x}^{\text{gt}}, \mathbf{x}(\mathbf{z}^*)\right) \\ &\text{s.t. } \mathbf{z}^* = \arg\min_{\mathbf{z}} t \cdot \mathbf{c}(\mathbf{f}, \boldsymbol{\Theta})^\top \mathbf{x}(\mathbf{z}) + P(\mathbf{x}(\mathbf{z})), \end{aligned} \tag{5}$$

which is now well-defined. We are interested in computing the gradient of the loss $\mathcal{L}$ w.r.t. the parameters $\boldsymbol{\Theta}$ of our cost function $\mathbf{c}(\cdot, \boldsymbol{\Theta})$. It is sufficient to show $\frac{\partial \mathcal{L}}{\partial \mathbf{c}}$, as gradients for the parameters $\boldsymbol{\Theta}$ can be obtained via the chain rule assuming $\mathbf{c}(\cdot; \boldsymbol{\Theta})$ is differentiable w.r.t. $\boldsymbol{\Theta}$.

The basic idea for computing gradients of problem (5) is to make use of implicit differentiation on the optimality condition of the lower level problem. For an uncluttered notation, we drop all dependencies of functions in the following. We define the desired gradient via chain rule as

$$\frac{\partial \mathcal{L}}{\mathbf{c}} = \frac{\partial \mathbf{z}^*}{\partial \mathbf{c}} \frac{\partial \mathbf{x}}{\partial \mathbf{z}^*} \frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \frac{\partial \mathbf{z}^*}{\partial \mathbf{c}} \mathbf{B}^\top \frac{\partial \mathcal{L}}{\partial \mathbf{x}}. \tag{6}$$

We assume the loss function $\mathcal{L}$ to be differentiable w.r.t. $\mathbf{x}$. To compute $\frac{\partial \mathbf{z}^*}{\partial \mathbf{c}}$, we use the optimality condition of (4)

$$\begin{aligned} \mathbf{0} &= \frac{\partial}{\partial \mathbf{z}} \left[t \cdot \mathbf{c}^\top \mathbf{x} + P \right] \\ &= t \cdot \frac{\partial \mathbf{x}}{\partial \mathbf{z}} \mathbf{c} + \frac{\partial \mathbf{x}}{\partial \mathbf{z}} \frac{\partial P}{\partial \mathbf{x}} = t \cdot \mathbf{B}^\top \mathbf{c} + \mathbf{B}^\top \frac{\partial P}{\partial \mathbf{x}} \end{aligned} \tag{7}$$

and differentiate w.r.t. $\mathbf{c}$, which gives

$$\begin{aligned} \mathbf{0} &= \frac{\partial}{\partial \mathbf{c}} \left[t \cdot \mathbf{B}^\top \mathbf{c} \right] + \frac{\partial}{\partial \mathbf{c}} \left[\mathbf{B}^\top \frac{\partial P}{\partial \mathbf{x}} \right] \\ &= t \cdot \mathbf{B} + \frac{\partial \mathbf{z}}{\partial \mathbf{c}} \frac{\partial \mathbf{x}}{\partial \mathbf{z}} \frac{\partial^2 P}{\partial \mathbf{x}^2} \mathbf{B} = t \cdot \mathbf{B} + \frac{\partial \mathbf{z}}{\partial \mathbf{c}} \mathbf{B}^\top \frac{\partial^2 P}{\partial \mathbf{x}^2} \mathbf{B} \end{aligned} \tag{8}$$

and which can be rearranged to

$$\frac{\partial \mathbf{z}}{\partial \mathbf{c}} = -t \cdot \mathbf{B} \left[\mathbf{B}^\top \frac{\partial^2 P}{\partial \mathbf{x}^2} \mathbf{B} \right]^{-1}. \tag{9}$$

The final derivative can then be written as

$$\frac{\partial \mathcal{L}}{\mathbf{c}} = -t \cdot \mathbf{B} \left[\mathbf{B}^\top \frac{\partial^2 P}{\partial \mathbf{x}^2} \mathbf{B} \right]^{-1} \mathbf{B}^\top \frac{\partial \mathcal{L}}{\partial \mathbf{x}}. \tag{10}$$

To fully define (10), we provide the second derivative of P w.r.t. $\mathbf{x}$, which is given as

$$\frac{\partial^2 P}{\partial \mathbf{x}^2} = \frac{\partial^2 P}{\partial \mathbf{x} \partial \mathbf{x}^\top} = \sum_{i=1}^{2M} \frac{1}{\left(b_i - \mathbf{a}_i^\top \mathbf{x}\right)^2} \cdot \mathbf{a}_i \mathbf{a}_i^\top. \tag{11}$$

In the supplemental material we show that (10) is equivalent to a generic solution provided in [36] and that $\mathbf{B}^\top \frac{\partial^2 P}{\partial \mathbf{x}^2} \mathbf{B}$ is always invertible.

3.2.3 Discussion

Training requires to solve the smoothed linear program (4), which can be done with any convex solver. This is essentially one step in a path-following method with a fixed temperature t. As suggested in [5], we set $t = \frac{M}{\epsilon}$, where ϵ is a hyper-parameter defining the approximation accuracy of the log barriers. We tried different values for ϵ and also an annealing scheme, but the results seem insensitive to this choice. We found $\epsilon = 0.1$ to work well in practice.

It is also important to note that our formulation is not limited to the task of MOT. It can be employed for any application where it is desirable to learn costs functions from data for an association problem, or, more generally, for a linear program with the assumptions given in Section 3.2.1. Our formulation can also be interpreted as one particular layer in a neural network that solves a linear program. The analogy between solving the smoothed linear program (4) and computing the gradients (10) with the forward and backward pass of a layer in a neural network is illustrated in Figure 2.

3.3. Defining ground truth and the loss function

To learn the parameters $\boldsymbol{\Theta}$ of the cost functions we need to compare the LP solution $\mathbf{x}^*$ with the ground truth solution $\mathbf{x}^{\text{gt}}$ in a loss function $\mathcal{L}$. Basically, $\mathbf{x}^{\text{gt}}$ defines which edges in the network flow graph should be active ($x_i^{\text{gt}} = 1$) and inactive ($x_i^{\text{gt}} = 0$). Training data needs to contain the ground truth bounding boxes (with target identities) and the detection bounding boxes. The detections define the structure of the network flow graph (see Section 3.1).

To generate $\mathbf{x}^{\text{gt}}$, we first match each detection with ground truth boxes in each frame individually. Similar to the evaluation of object detectors, we match the highest scoring detection having an intersection-over-union overlap larger 0.5 to each ground truth bounding box. This divides the set of detection into true and false positives and already defines the ground truth for x^{det}. In order to provide ground truth for associations between detections, *i.e.*, x^{link}, we iterate the frames sequentially and investigate all edges pointing forward in time for each detection. We activate the edge that points to the closest true positive detection in time, which has the same target identity. All other x^{link} edges are set to 0. After all ground truth trajectories are identified, it is straightforward to set the ground truth of x^{in} and x^{out}.

As already pointed out in [50], there exist different types of links that should be treated differently in the loss function. There are edges x^{link} between two false positives (FP-FP), between true and false positives (TP-FP), and between

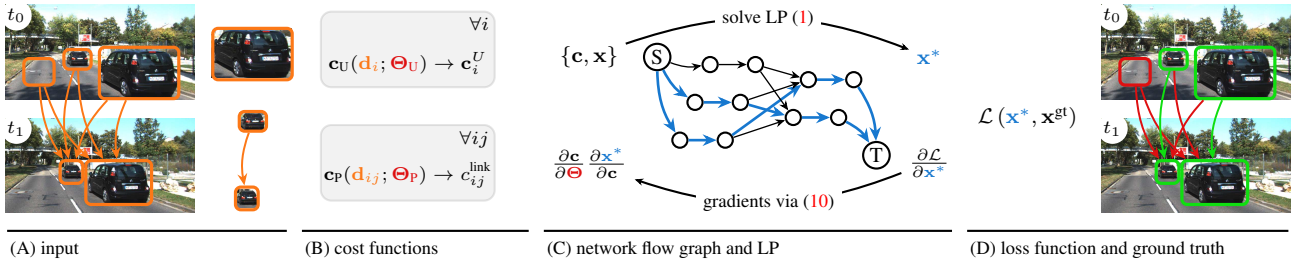

Figure 2: During inference, two cost functions (B) predict unary and pair-wise costs based on features extracted from detections on the input frames (A). The costs drive the network flow (C). During training, a loss compares the solution $\mathbf{x}^*$ with ground truth $\mathbf{x}^{\mathrm{gt}}$ to back-propagate gradients to the parameters $\boldsymbol{\Theta}$.

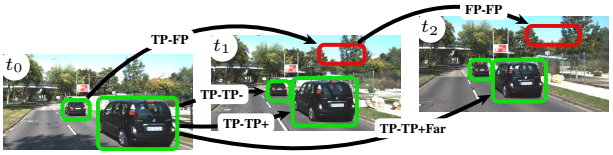

Figure 3: An illustration of different types of links that emerge when computing the loss. See text for more details on the different combinations of true (TP, green) and false positive (FP, red) detections.

two true positives with the same (TP-TP+) or a different (TP-TP-) identity. For (TP-TP+) links, we also differentiate between the shortest links for the trajectory and links that are longer (TP-TP+Far). Edges associated with a single detection (x^{in}, x^{det} and x^{out}) are either true (TP) or false positives (FP). Figure 3 illustrates all these cases. To trade-off the importance between these types, we define the following weighted loss function

$$\mathcal{L}(\mathbf{x}^*, \mathbf{x}^{\mathrm{gt}}) = \sum_{\kappa \in \{\mathrm{in,det,out}\}} \sum_i \omega_i (x_i^{\kappa,*} - x_i^{\mathrm{gt}})^2 + \sum_{i,j \in \mathcal{E}} \omega_{ij} (x_{i,j}^{\mathrm{link},*} - x_{i,j}^{\mathrm{gt}})^2, \tag{12}$$

where $\mathcal{E}$ is the set of all edges between detections i and j. Note that the weights can be adjusted for each variable separately. The default value for the weights is 1, but we can adjust them to incorporate three intuitions about the loss. (i) Ambiguous edges: Detections of an (FP-FP) link may describe a consistently tracked but wrong object. Also, detections of a (TP-TP+Far) link are obviously very similar. In both cases the ground truth variable is still inactive. It may hurt the learning procedure if a wrong prediction is penalized too much for these cases. Thus, we can set $\omega_{i,j} = \omega_{\mathrm{amb}} < 1$. (ii) To influence the trade-off between precision and recall, we define the weight ω_{pr} for all edges involving a true positive detection. Increasing ω_{pr} favors recall. (iii) To emphasize associations, we additionally weight all x^{link} variables with ω_{link}. If multiple of these cases are true for a single variable, we multiply the weights.

Finally, we note that [50] uses a different weighting scheme and an ℓ_1 loss. We compare this definition with various weightings of our loss function in Section 4.3.

3.4. Tracking model

After the training phase, the above described network flow formulation can be readily applied for tracking. One option is to batch process whole sequences at once, which, however, does not scale to long sequences. Lenz *et al*. [30] present a sophisticated approximation with bounded memory and computation costs. As we focus on the learning phase in this paper, we opt for a simpler approach, which empirically gives similar results to batch processing but does not come with guarantees as in [30].

We use a temporal sliding window of length W that breaks a video sequence into chunks. We solve the LP problem for the frames inside the window, move it by Δ frames and solve the new LP problem, where $0 < \Delta < W$ ensures a minimal overlap of the two solutions. Each solution contains a separate set of trajectories, which we associate with bipartite graph matching to carry the object identity information over time. The matching cost for each pair of trajectories is inversely proportional to the number of detections they share. Unmatched trajectories get new identities.

In practice, we use maximal overlap, *i.e*., $\Delta = 1$, to ensure stable associations of trajectories between two LP solutions. For each window, we output the detections of the middle frame, *i.e*., looking $\frac{W}{2}$ frames into future and past, similar to [10]. Note that using detections from the latest frame as output enables on-line processing.

4. Experiments

To evaluate the proposed tracking algorithm we use the publicly available benchmarks KITTI tracking [17], MOT15 [28] and MOT16 [32]. The data sets provide training sets of 21, 11 and 7 sequences, respectively, which are fully annotated. As suggested in [17, 28, 32], we do a (4-fold) cross validation for all our experiments, except for the benchmark results in Section 4.4.

To assess the performance of the tracking algorithms we rely on standard MOT metrics, CLEAR MOT [4] and MT/PT/ML [31], which are also used by both benchmarks [17, 28]. This set of metrics measures recall and precision, both on a detection and trajectory level, counts the number of identity switches and fragmentations of trajectories and also provides an overall tracking accuracy (MOTA).

4.1. Learned versus hand-crafted cost functions

The main contribution of this paper is a novel way to automatically learn parameterized cost functions for a network flow based tracking model from data. We illustrate the efficacy of the learned cost functions by comparing them with two standard choices for hand-crafted costs. First, we follow [29] and define $c_i^{\text{det}} = \log(1 - p(\mathbf{d}_i))$, where $p(\mathbf{d}_i)$ is the detection probability, and

$$c_{i,j}^{\text{link}} = -\log \text{E}\left(\frac{\|\mathbf{b}(\mathbf{d}_i) - \mathbf{b}(\mathbf{d}_j)\|}{\Delta_t}, V_{\text{max}}\right) - \log(B^{\Delta_t - 1}), \tag{13}$$

where $\text{E}(V_t, V_{\text{max}}) = \frac{1}{2} + \frac{1}{2}\text{erf}(\frac{-V_t + 0.5 \cdot V_{\text{max}}}{0.25 \cdot V_{\text{max}}})$ with $\text{erf}(\cdot)$ being the Gauss error function and Δ_t is the frame difference between i and j. While [29] defines a slightly different network flow graph, we keep the graph definition the same (see Section 3.1) for all methods to ensure a fair comparison of the costs. Second, we hand-craft our own cost function and define $c_i^{\text{det}} = \alpha \cdot p(\mathbf{d}_i)$ as well as

$$c_{i,j}^{\text{link}} = (1 - \text{IoU}(\mathbf{b}(\mathbf{d}_i), \mathbf{b}(\mathbf{d}_j))) + \beta \cdot (\Delta_t - 1), \tag{14}$$

where $\text{IoU}(\cdot, \cdot)$ is the intersection over union. We tune all parameters, *i.e.*, $c_i^{\text{in}} = c_i^{\text{out}} = C$ (we did not observe any benefit when choosing these parameters separately), B, V_{max}, α and β, with grid search to maximize MOTA while balancing recall. Note that the exponential growth of the search space w.r.t. the number of parameters makes grid search infeasible at some point.

With the same source of input information, *i.e.*, bounding boxes $\mathbf{b}(\mathbf{d})$ and detection confidences $p(\mathbf{d})$, we train various types of parameterized functions with the algorithm proposed in Section 3.2. For unary costs, we use the same parameterization as for the hand-crafted model, *i.e.*, constants for c^{in} and c^{out} and a linear model for c^{det}. However, for the pair-wise costs, we evaluate a linear model, a one-layer MLP with 64 hidden neurons and a two-layer MLP with 32 hidden neurons in both layers. The input feature $\mathbf{f}$ is the difference between the two bounding boxes, their detection confidences, the normalized time difference, as well as the IoU value. We train all three models for 50k iterations using ADAM [24] with a learning rate of 10^{-4}, which we decrease by a factor of 10 every 20k iterations.

Table 1 shows that our proposed training algorithm can successfully learn cost functions from data on both KITTI-Tracking and MOT16 data sets. With the same input information given, our approach even slightly outperforms both hand-crafted baselines in terms of MOTA. In particular, we observe lower identity switches and fragmentations on KITTI-Tracking and higher recall and mostly-tracked on MOT16. While our hand-crafted function (14) is inherently limited when objects move fast and IoU becomes 0 (compared to (13) [29]), both still achieve similar performance. For both baselines, we did a hierarchical grid search to get good results. However, an even finer grid search would be required to achieve further improvements. The attraction of our method is that it obviates the need for such a tedious search and provides a principled way of finding good parameters. We can also observe from the tables that non-linear functions (MLP 1 and MLP 2) perform better than linear functions (Linear), which is not possible in [49].

	MOTA	REC	PREC	MT	IDS	FRAG
Crafted [29]	73.64	83.54	92.99	58.73	121	459
Crafted-ours	73.75	83.92	92.65	59.44	89	431
Linear	73.51	83.47	92.99	59.08	132	430
MLP 1	74.09	83.93	92.87	59.61	70	371
MLP 2	74.19	84.07	92.85	59.96	70	376

(a)

	MOTA	REC	PREC	MT	IDS	FRAG
Crafted [29]	28.28	29.94	95.04	5.80	111	1063
Crafted-ours	29.19	34.01	87.88	6.77	142	1272
Linear	28.25	38.01	80.09	9.67	342	1620
MLP 1	31.05	37.51	85.81	8.32	282	1553
MLP 2	31.10	37.53	85.88	8.51	289	1562

(b)

Table 1: Learned vs. hand-crafted cost functions on a cross-validation on (a) KITTI-Tracking [17] and (b) MOT16 [32].

4.2. Combining multiple input sources

Recent works have shown that temporal and appearance features are often beneficial for MOT. Choi [10] presents a spatio-temporal feature (ALFD) to compare two detections, which summarizes statistics from tracked interest points in a 288-dimensional histogram. Leal-Taixé *et al*. [27] show how to use raw RGB data with a Siamese network to compute an affinity metric for pedestrian tracking. Incorporating such information into a tracking model typically requires (i) an isolated learning phase for the affinity metric and (ii) some hand-tuning to combine it with other affinity metrics and other costs in the model (*e.g.*, c^{in}, c^{det}, c^{out}). In the following, we demonstrate the use of both motion and appearance features in our framework.

Motion-features: In Table 2, we demonstrate the impact of the motion feature ALFD [10] compared to purely spatial features on the KITTI-Tracking data set as in [10]. For each source of input, we compare both hand-crafted (C) and learned (L) pair-wise cost functions. First, we use

Inputs	MOTA	REC	PREC	MT	IDS	FRAG
(C) B	73.64	83.54	92.99	58.73	121	459
(L) B	73.65	84.55	92.00	61.55	89	422
(C) B+O	73.75	83.92	92.65	59.44	89	431
(L) B+O	74.12	84.13	92.69	60.49	55	361
(C) B+O+M	73.07	85.07	90.92	61.73	43	386
(L) B+O+M	74.11	84.74	92.05	61.73	29	335

Table 2: We evaluate the influence of different types of input sources, raw detection inputs (B), bounding box overlaps (O) and the ALFD motion feature [10] (M) for both learned (L) and hand-crafted (C) costs on KITTI-Tracking [17].

only the raw bounding box information (B), *i.e.*, location and temporal difference and detection score. For the hand-crafted baseline, we use the cost function defined in (13), *i.e.*, [29]. Second, we add the IoU overlap (B+O) and use (14) for the hand-crafted baseline. Third, we incorporate ALFD [10] into the cost (B+O+M). To build a hand-crafted baseline for (B+O+M), we construct a separate training set of ALFD features containing examples for positive and negative matches and train an SVM on the binary classification task. During tracking, the normalized SVM scores $\hat{s}_{\text{A}}$ (a sigmoid function maps the raw SVM scores into $[0,1]$) are incorporated into the cost function

$$c_{i,j}^{\text{link}} = (1-\text{IoU}(\mathbf{b}(\mathbf{d}_i),\mathbf{b}(\mathbf{d}_j)))+\beta\cdot(\Delta_t-1)+\gamma\cdot(1-\hat{s}_{\text{A}}), \quad (15)$$

where γ is another hyper-parameter we also tune with grid-search. For our learned cost functions, we use a 2-layer MLP with 64 neurons in each layer to predict $c_{i,j}^{\text{link}}$ for the (B) and (B+O) options. For (B+O+M), we use a separate 2-layer MLP to process the 288-dimensional ALFD feature, concatenate both 64-dimensional hidden vectors of the second layers, and predict $c_{i,j}^{\text{link}}$ with a final linear layer.

Table 2 again shows that learned cost functions outperform hand-crafted costs for all input sources, which is consistent with the previous experiment in Section 4.1. The table also demonstrates the ability of our approach to make effective use of the ALFD motion feature [10], especially for identity switches and fragmentations. While it is typically tedious and suboptimal to combine such diverse features in hand-crafted cost functions, it is easy with our learning method because all parameters can still be jointly trained under the same loss function.

Appearance features: Here, we investigate the impact of raw RGB data on both unary and pair-wise costs of the network flow formulation. We use the MOT15 data set [28] and the provided ACF detections [14]. First, we integrate the raw RGB data into the unary cost c_i^{det} (Au). For each detected bounding box $\mathbf{b}(\mathbf{d}_i)$, we crop the underlying RGB patch I_i with a fixed aspect ratio, resize the patch to 128×64

Unary cost	MOTA	REC	PREC	MT	IDS	FRAG
Crafted [29]	30.55	38.54	83.70	11.60	194	853
Crafted-ours	30.43	38.98	82.69	11.40	156	825
(B+O)	28.94	43.63	75.47	14.00	204	962
Au+(B+O)	39.08	46.99	86.71	15.60	285	1062
Au+(B+O+Ap)	39.23	47.17	86.50	15.80	233	954

Table 3: Using appearance for unary (Au) and pair-wise (Ap) cost functions clearly improves tracking performance.

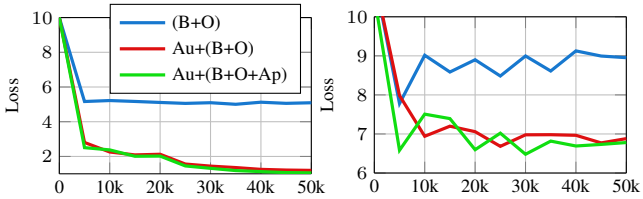

Figure 4: The difference in the loss on the training (left) and validation set (right) over 50k iterations of training for models w/ (Au,Ap) and w/o appearance features.

and define the cost

$$c_i^{\text{det}} = \mathbf{c}_{\text{conf}}(p(\mathbf{d}_i);\boldsymbol{\Theta}_{\text{conf}}) + \mathbf{c}_{\text{Au}}(I_i;\boldsymbol{\Theta}_{\text{Au}}), \quad (16)$$

which consists of one linear function taking the detection confidence and one deep network taking the image patch. We choose ResNet-50 [19] to extract features for $\mathbf{c}_{\text{Au}}$ but any other differentiable function can be used as well.

Second, we use a Siamese network (same as for unary term) that compares RGB patches of two detections, similar to [27] but without optical flow information. As with the motion features above, we use a two-stream network to combine spatial information (B+O) with appearance features (Ap). The hidden feature vector of a 2-layer MLP (B+O) is concatenated with the difference of the hidden features from the Siamese network. A final linear layer predicts the costs $c_{i,j}^{\text{link}}$ of the pair-wise terms.

Table 3 shows that integrating RGB information into the detection cost Au+(B+O) improves tracking performance significantly over the baselines. Using the RGB information in the pair-wise cost as well Au+(B+O+Ap) further improves results, especially for identity switches and fragmentations. Figure 4 visualizes the loss on the training and validation set for the three learning-based methods, which again shows the impact of appearance features. Note, however, that the improvement is limited because we still rely on the underlying ACF detector and are not able to improve recall over the recall of the detector. But the experiment clearly shows the potential ability to integrate deep network based object detectors directly into an end-to-end tracking framework. We plan to investigate this avenue in future work.

Weighting	MOTA	REC	PREC	MT	IDS	FRAG
none	74.07	82.84	93.78	57.67	53	333
[49]	73.99	82.90	93.63	57.32	43	331
none-ℓ_1	73.90	83.43	93.17	58.73	77	362
[49]-ℓ_1	73.92	83.19	93.38	58.73	71	357
$\omega_{\text{basic}} = 0.1$	74.15	84.11	92.72	60.49	51	360
$\omega_{\text{basic}} = 0.5$	74.13	83.90	92.92	59.96	62	363
$\omega_{\text{pr}} = 0.3$	66.84	70.68	98.35	28.92	34	216
$\omega_{\text{pr}} = 1.5$	73.28	85.52	90.85	63.49	80	387
$\omega_{\text{links}} = 1.5$	74.14	84.53	92.31	61.38	45	357
$\omega_{\text{links}} = 2.0$	74.10	84.80	92.03	61.38	42	358

Table 4: Differently weighting the loss function provides a trade-off between various behaviors of the learned costs.

4.3. Weighting the loss function

For completeness, we also investigate the impact of different weighting schemes for the loss function defined in Section 3.3. First, we compare our loss function without any weighting (none) with the loss defined in [49]. We also do this for an ℓ_1 loss. We can see from the first part in Table 4 that both achieve similar results but [49] achieves slightly better identity switches and fragmentations. By decreasing ω_{basic} we limit the impact of ambiguous cases (see Section 3.3) and can observe a slight increase in recall and mostly tracked. Also, we can influence the trade-off between precision and recall with ω_{pr} and we can lower the number of identity switches by increasing ω_{links}.

4.4. Benchmark results

Finally, we evaluate our learned cost functions on the benchmark test sets. For KITTI-Tracking [17], we train cost functions equal to the ones described in Section 4.2 with ALFD motion features [10], *i.e.*, (B+O+M) in Table 2. We train the models on the full training set and upload the results on the benchmark server. Table 5 compares our method with other off-line approaches that use RegionLet detections [51]. While [10] achieves better results on the benchmark, their approach includes a complex graphical model and a temporal model for trajectories. The fair comparison is with Wang and Fowlkes [50], which is the most similar approach to ours. While we achieve better MOTA, it is important to note that the comparison needs to be taken with a grain of salt. We include motion features in the form of ALFD [10]. On the other hand, the graph in [50] is more complex as it also accounts for trajectory interactions.

We also evaluate on the MOT15 data set [28], where we choose the model that integrates raw RGB data into the unary costs, *i.e.*, Au+(B+O) in Table 3. We achieve an MOTA value of 26.8, compared to 25.2 for [50] (most similar model) and 29.0 for [27] (using RGB data for pair-wise term). We again note that [27] additionally integrates optical flow into the pair-wise term. The impact of RGB features is not as pronounced as in our cross-validation experiment in Table 3. The most likely reason we found for this scenario is over-fitting of the unary terms.

Method	MOTA	MOTP	MT	ML	IDS	FRAG
[30]	60.84	78.55	53.81	7.93	191	966
[10]	69.73	79.46	56.25	12.96	36	225
[34]	55.49	78.85	36.74	14.02	323	984
[50]	66.35	77.80	55.95	8.23	63	558
Ours	67.36	78.79	53.81	9.45	65	574

Table 5: Results on KITTI-Tracking [17] from 11/04/16.

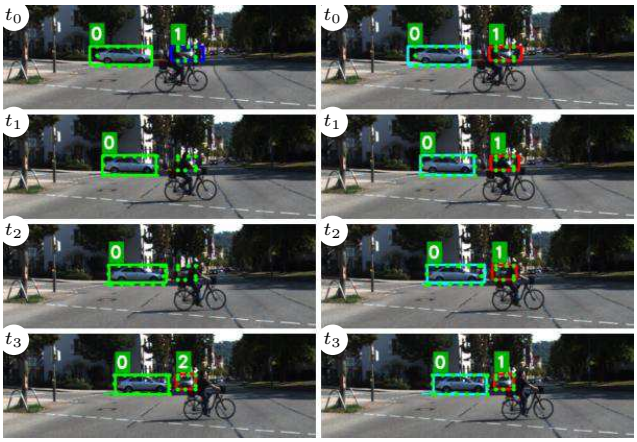

Figure 5: A qualitative example showing a failure case of the hand-crafted costs (left) compared to the learned costs (right), which leads to a fragmentation. The green dotted boxes are ground truth, the solid colored are ones tracked objects. The numbers are the object IDs. Best viewed in color and zoomed.

Figure 5 also gives a qualitative comparison between hand-crafted and learned cost functions on KITTI [17]. The supplemental material contains more qualitative results.

5. Conclusion

Our work demonstrates how to learn a parameterized cost function of a network flow problem for multi-object tracking in an end-to-end fashion. The main benefit is the gained flexibility in the design of the cost function. We only assume it to be parameterized and differentiable, enabling the use of powerful neural network architectures. Our formulation learns the costs of *all* variables in the network flow graph, avoiding the delicate task of hand-crafting these costs. Moreover, our approach also allows for easily combining different sources of input data. Evaluations on three public data sets confirm these benefits empirically.

For future works, we plan to integrate object detectors end-to-end into this tracking model, investigate more complex network flow graphs with trajectory interactions and explore applications to max-flow problems.

References

[1] A. Andriyenko, K. Schindler, and S. Roth. Discrete-Continuous Optimization for Multi-Target Tracking. In *CVPR*, 2012. 2

[2] S.-H. Bae and K.-J. Yoon. Robust Online Multi-Object Tracking based on Tracklet Confidence and Online Discriminative Appearance Learning. In *CVPR*, 2014. 2

[3] J. Berclaz, F. Fleuret, E. Türetken, and P. Fua. Multiple Object Tracking using K-Shortest Paths Optimization. *PAMI*, 33(9):1806–1819, 2011. 1, 2

[4] K. Bernardin and R. Stiefelhagen. Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics. *EURASIP Journal on Image and Video Processing*, 2008. 6

[5] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. 3, 4

[6] J. Bracken and J. T. McGill. Mathematical Programs with Optimization Problems in the Constraints. *Operations Research*, 21:37–44, 1973. 1, 2

[7] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool. Robust Tracking-by-Detection using a Detector Confidence Particle Filter. In *ICCV*, 2009. 1, 2

[8] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool. Online Multiperson Tracking-by-Detection from a Single, Uncalibrated Camera. *PAMI*, 33(9):1820–1833, 2011. 2

[9] A. A. Butt and R. T. Collins. Multi-target Tracking by Lagrangian Relaxation to Min-Cost Network Flow. 2013. 2

[10] W. Choi. Near-Online Multi-target Tracking with Aggregated Local Flow Descriptor. In *ICCV*, 2015. 1, 2, 3, 5, 6, 7, 8

[11] W. Choi, C. Pantofaru, and S. Savarese. A General Framework for Tracking Multiple People from a Moving Camera. *PAMI*, 35(7):1577–1591, 2013. 2

[12] W. Choi and S. Savarese. A Unified Framework for Multi-Target Tracking and Collective Activity Recognition. In *ECCV*, 2012. 2

[13] B. Colson, P. Marcotte, and G. Savard. An overview of bilevel optimization. *Annals of Operations Research*, 153(1):235–256, 2007. 1, 2

[14] P. Dollár, R. Appel, S. Belongie, and P. Perona. Fast Feature Pyramids for Object Detection. *PAMI*, 36(8):1532–1545, 2014. 7

[15] A. Ess, B. Leibe, K. Schindler, and L. van Gool. Robust Multi-Person Tracking from a Mobile Platform. *PAMI*, 31(10):1831–1846, 2009. 1, 2

[16] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object Detection with Discriminatively Trained Part Based Models. *PAMI*, 32(9):1627–1645, 2010. 1

[17] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *CVPR*, 2012. 1, 5, 6, 7, 8

[18] N. J. Gordon, D. Salmond, and A. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings F (Radar and Signal Processing)*, 140:107–113, 1993. 2

[19] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016. 7

[20] C. Huang, B. Wu, and R. Nevatia. Robust Object Tracking by Hierarchical Association of Detection Responses. In *ECCV*, 2008. 2

[21] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960. 2

[22] Z. Khan, T. Balch, and F. Dellaert. MCMC-Based Particle Filtering for Tracking a Variable Number of Interacting Targets. *PAMI*, 27(11):1805–1819, 2005. 1

[23] S. Kim, S. Kwak, J. Feyereisl, and B. Han. Online Multi-Target Tracking by Large Margin Structured Learning. In *ACCV*, 2012. 2

[24] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015. 6

[25] H. W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955. 1, 2

[26] C.-H. Kuo, C. Huang, and R. Nevatia. Multi-Target Tracking by On-Line Learned Discriminative Appearance Models. In *CVPR*, 2010. 1, 2

[27] L. Leal-Taixé, C. Canton-Ferrer, and K. Schindler. Learning by tracking: Siamese CNN for robust target association. In *DeepVision: Deep Learning for Computer Vision, CVPR Workshop*, 2016. 1, 2, 6, 7, 8

[28] 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. 1, 5, 6, 7, 8

[29] L. Leal-Taixé, G. Pons-Moll, and B. Rosenhahn. Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker. 2011. 1, 2, 6, 7

[30] P. Lenz, A. Geiger, and R. Urtasun. FollowMe: Efficient Online Min-Cost Flow Tracking with Bounded Memory and Computation. In *ICCV*, 2015. 2, 5, 8

[31] Y. Li, C. Huang, and R. Nevatia. Learning to Associate: HybridBoosted Multi-Target Tracker for Crowded Scene. In *CVPR*, 2009. 1, 2, 6

[32] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler. MOT16: A Benchmark for Multi-Object Tracking. *arXiv:1603.00831*, 2016. 1, 5, 6

[33] A. Milan, S. Roth, and K. Schindler. Continuous Energy Minimization for Multitarget Tracking. *PAMI*, 36(1):58–72, 2014. 1, 2

[34] A. Milan, K. Schindler, and S. Roth. Detection- and Trajectory-Level Exclusion in Multiple Object Tracking. In *CVPR*, 2013. 8

[35] J. Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957. 1, 2

[36] P. Ochs, R. Ranftl, T. Brox, and T. Pock. Bilevel Optimization with Nonsmooth Lower Level Problems. In *SSVM*, 2015. 4

[37] S. Oh, S. Russell, and S. Sastry. Markov Chain Monte Carlo Data Association for Multiple-Target Tracking. *IEEE Transactions on Automatic Control*, 54(3):481–497, 2009. 1

[38] K. Okuma, A. Taleghani, N. De Freitas, J. J. Little, and D. G. Lowe. A Boosted Particle Filter: Multitarget Detection and Tracking. In *ECCV*, 2004. 1

[39] S. Pellegrini, A. Ess, K. Schindler, and L. van Gool. Youll Never Walk Alone: Modeling Social Behavior for Multi-target Tracking. In *ICCV*, 2009. 2

[40] H. Pirsiavash, D. Ramanan, and C. C. Fowlkes. Globally-Optimal Greedy Algorithms for Tracking a Variable Number of Objects. In *CVPR*, 2011. 2

[41] H. Possegger, T. Mauthner, P. M. Roth, and H. Bischof. Occlusion Geodesics for Online Multi-Object Tracking. In *CVPR*, 2014. 1, 2

[42] R. Ranftl and T. Pock. A Deep Variational Model for Image Segmentation. In *GCPR*, 2014. 2

[43] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 1

[44] G. Riegler, R. Ranftl, M. Rüther, T. Pock, and H. Bischof. Depth Restoration via Joint Training of a Global Regression Model and CNNs. In *BMVC*, 2015. 2

[45] G. Riegler, M. Rüther, and H. Bischof. ATGV-Net: Accurate Depth Super-Resolution. In *ECCV*, 2016. 2

[46] F. Solera, S. Calderara, and R. Cucchiara. Learning to Divide and Conquer for Online Multi-Target Tracking. In *CVPR*, 2015. 2

[47] X. Song, J. Cui, H. Zha, and H. Zhao. Vision-based Multiple Interacting Targets Tracking via On-line Supervised Learning. In *ECCV*, 2008. 2

[48] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large Margin Methods for Structured and Interdependent Output Variables. *JMLR*, 6:1453–1484, 2005. 2

[49] S. Wang and C. Fowlkes. Learning Optimal Parameters For Multi-target Tracking. In *BMVC*, 2015. 1, 2, 6, 8

[50] S. Wang and C. Fowlkes. Learning Optimal Parameters for Multi-target Tracking with Contextual Interactions. *IJCV*, pages 1–18, 2016. 1, 4, 5, 8

[51] X. Wang, M. Yang, S. Zhu, and Y. Lin. Regionlets for Generic Object Detection. In *ICCV*, 2013. 1, 8

[52] Y. Xiang, A. Alahi, and S. Savarese. Learning to Track: Online Multi-Object Tracking by Decision Making. In *ICCV*, 2015. 1

[53] A. R. Zamir, A. Dehghan, and M. Shah. GMCP-Tracker: Global Multi-object Tracking Using Generalized Minimum Clique Graphs. In *ECCV*, 2012. 2

[54] L. Zhang, Y. Li, and R. Nevatia. Global Data Association for Multi-Object Tracking Using Network Flows. In *CVPR*, 2008. 1, 2, 3