This ICCV Workshop paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Multi Target Tracking from Drones by Learning from Generalized Graph Differences

Håkan Ardö
Axis Communications
hakanad@axis.com

Mikael Nilsson
Center for Mathematical Sciences
Lund University
micken@maths.lth.se

Abstract

Formulating the multi object tracking problem as a network flow optimization problem is a popular choice. The weights of such network flow problem can be learnt efficiently from training data using a recently introduced concept called Generalized Graph Differences (GGD). This allows a general tracker implementation to be specialized to drone videos by training it on the VisDrone dataset. Two modifications to the original GGD is introduced in this paper and a result with an average precision of 23.09 on the test set of VisDrone 2019 was achieved.

1. Introduction

Single frame object detectors have recently become very powerful [18, 14, 7, 13, 6, 23]. These will for each frame in a video give a list of objects in the scene and for each object its class (person, car, ...) and some representation of the object location (bounding box, keypoints, pixel mask, ...). These detections can then be connected from frame to frame into object tracks using a multi target tracking algorithm. The task of that algorithm is also to discard false detections and fill in missing detections.

Network flow-based methods is a classical approach to resort to in multi target tracking [11, 17, 4, 25, 9, 2]. They can be computational efficient implemented [1] and it is often possible to guarantee a globally optimal solution even in an online setting [12].

In this paper a network flow tracker [2] using optical flow to observe the motion (instead of estimating it from observed positions) is extended and evaluated on the VisDrone dataset. Long range connections are utilizes in the flow graph. That allows the full tracking problem to be solved with a single optimization without the need to first produce tracklets that are later combined into tracks, which is often otherwise needed [20, 25].

The tracker embeds all feasible solutions (Eq 7) to the network flow problem into a one dimensional feature space consisting of a score. The aim is to make this score of the correct solution higher than the score of all other solutions. Then a linear program is used during inference to efficiently search for the highest scoring solution.

2. Tracking Algorithm

Here, an overview of the tracking algorithm [2] will be presented as a constrained mathematical optimisation problem. It can then be solved using either network flow algorithms or more general linear programming. To keep the formulation simple, only a single object class is considered. But generalizing to multiple classes is straight forward.

2.1. Basic graph formulation

The basic idea behind the algorithm is to build a graph with object detections as vertices and use sparse optical flow feature point tracks, KLT-tracks [24], to connect these vertices with edges. Then a flow capacity of one is assigned to each edge and a network flow problem is solved. The solution will have a positive flow of one between detections that belong to the same object, see Figure 1.

The input to the algorithm is a set of detections,

$$V = \left\{ v_0, v_1, \cdots v_{|V|} \right\}, \tag{1}$$

produced by an object detector. Each detection, $v_k = (t_k, L_k, c_k)$ consists of a frame number, t_k, a location, L_k and a confidence c_k. The location represents which pixels in the image the object covers. It is defined as a bounding-box and pixels, (x, y), located on the object are be denoted $(x, y) \in L_k$.

In addition to the detections there is also KLT-tracks consisting of a set of point tracks

$$P = \left\{ P_0, P_1, \cdots, P_{|P|} \right\}, \tag{2}$$

where $P_i = \left(p_{i,0}, p_{i,1}, \cdots p_{i,|P_i|} \right)$ and $p_{i,j} = (t_{i,j}, x_{i,j}, y_{i,j}, c_{i,j})$. Here $t_{i,j}$ is the global frame number

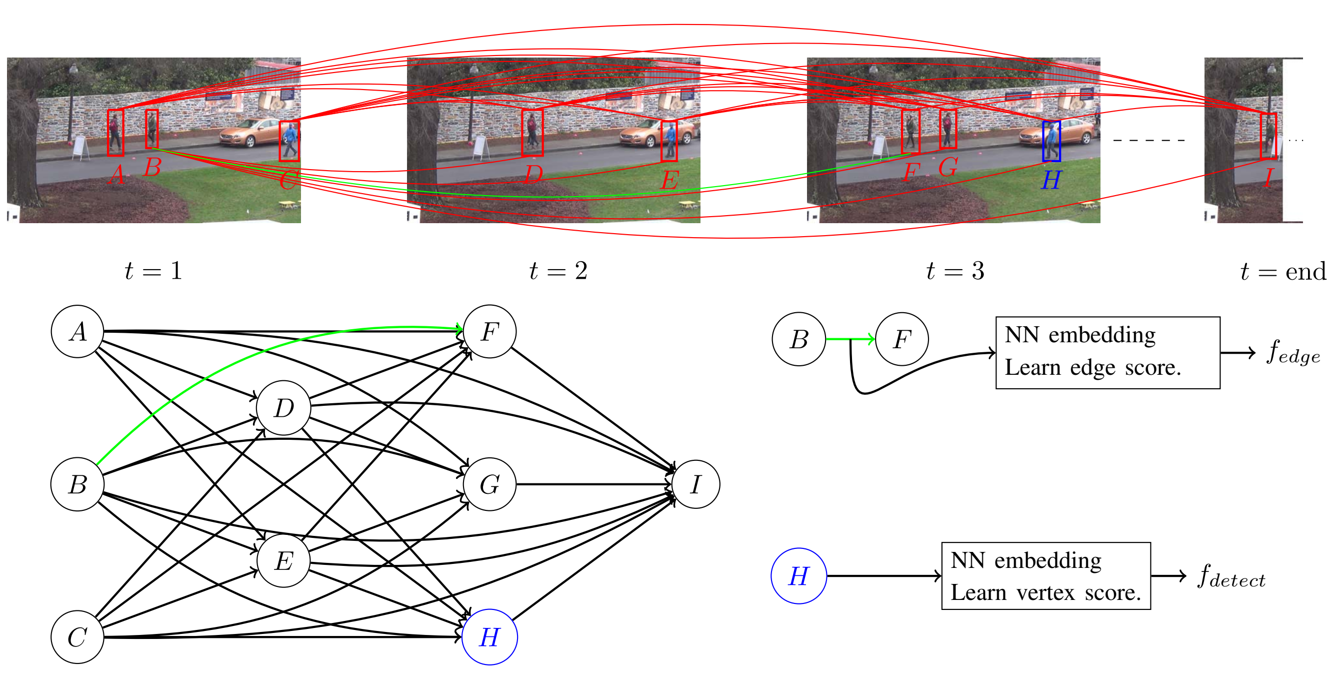


Figure 1. Concept of proposed method to address tracking with a graph and learning mapping for edges and vertices.

and $(x_{i,j}, y_{i,j})$ is the pixel location of the KLT-track in that frame and $c_{i,j}$ is a confidence. The confidence used here is the negated L1 distance between a small patch centered around the point in frame $t_{i,j}$ and $t_{i,j}$-1.

Each KLT-track will connect the detections it intersects into a sequences of detections. Each such sequence form one object track hypothesis. All of those hypothesis will be combined into edges in a graph representing different possible object tracks.

To formalize, a set A_i is introduced, that contains all detections intersecting the feature point track P_i,

$$A_i = \{ v_k \, | t_{i,j} = t_k, (x_{i,j}, y_{i,j}) \in L_k \quad \text{for some } j \} . \quad (3)$$

Then a graph is formed where the detections, v_k, are vertices and edges between the vertices are produced from neighbouring detections within each of the A_i tracks. Note that the distance between neighbouring detections in A_i might be several frames as the feature points can be tracked even if there are no detections. A neighbouring radius of $r_{\text{neighbours}}$ is used. That is, a detection is considered to be neighbour with the $r_{\text{neighbours}}$ preceding and $r_{\text{neighbours}}$ following detections. In order to avoid connecting distant detection a threshold, t_{max}, is introduced to discard such edges. That is, two detections, v_{k_1} and v_{k_2} are not considered neighbours if $|t_{k_2} - t_{k_1}| > t_{\text{max}}$. Also note that in the case of overlapping detections, A_i might contain two (or more) detections for the same frame. These detections are not considered neighbours to each other. Instead their neighbouring detections will have multiple incoming or outgoing edges. Formally, let $(v_{k_1}, v_{k_2}) \in N_i$ denote that v_{k_1} and v_{k_2} are neighbours in A_i according to the neighbouring structure described above. Then there is a set of directed edges,

$$E = \{ (v_{k_1}, v_{k_2}) \, | t_{k_2} > t_{k_1}, (v_{k_1}, v_{k_2}) \in N_i \quad \text{for some } i \} . \quad (4)$$

Each edge is weighted with a weight function, f_{edge}, that depend on all the KLT-tracks between the detections v_{k_1} and v_{k_2}, $P_{k_1,k_2} = \{ P_i \, | \, p_{i,j_1} \in L_{k_1}, p_{i,j_2} \in L_{k_2} \quad \text{for some } j_1, j_2 \}$. The vertexes are also weighted with a weight function, f_{detect}, that depend on the detection v_k. These are learned from training data, see Section 3.

2.2. Long range connections

To allow objects to occlude each other, long range connections are added to the graph. The problem is that during an occlusion a lot of feature point tracks will jump from one object to the other, which means that the feature point tracks are not reliably in such situations. In order to address this issue, the objects are assumed to approximately follow the linear motion model [15]. Therefore, a velocity, $w_{i,k}^{\text{pre}}$, is also estimated for each KLT-track, P_i, intersecting the detection v_k. It is produced by fitting a line to the n_{velest} most recent positions preceding t_k of that KLT-track. Using this velocity the location can be projected into the n_{project} closest future frames, and connections made to detections there. The weights of such connections will depend both on how well the future detection matches the predicted location and on how well the estimated velocities match. This kind of edges can skip over problematic situations entirely and instead match velocity and position of incoming and outgoing tracks. The velocity of the outgoing detections, $w_{i,k}^{\text{post}}$, is calculated from the n_{velest} KLT-track positions directly following the detection time t_k. This way the incoming velocity is estimated prior to the occlusion and the outgoing velocity is estimated after the occlusion. That means that neither of them should be affect too much by confusing KLT-tracks jumping target during the occlusion.

A set of long connections, C_{k_1,k_2}, connecting v_{k_1} and v_{k_2} will be formed, with one connection for each KLT-track intersecting v_{k_1} that started more than n_{velest} before t_{k_1}. Each of these connection will be based on different velocity estimates, w_{i,k_1}^{pre}. That is

$$C_{k_1,k_2} = \left\{ w_{i,k_1}^{\text{pre}} \,|p_{i,j} \in L_{k_1}, \quad \text{for some } j \geq n_{\text{velest}} \right\}. \tag{5}$$

Now the edge weight function, f_{edge} depend on both the KLT-tracks and the long connections, $f_{\text{edge}}\left(P_{k_1,k_2}, C_{k_1,k_2}, v_{k_1}, v_{k_2}\right)$.

2.3. Network flow

Using the constructed graph, the multi target tracking problem can be formulated as a network flow problem. Indicator variables, $\hat{v}_k \in \{0,1\}$, are introduced that indicates whether each detection is a true positive or a false positive. Also, indicator variables, $\hat{e}_{k_1,k_2} \in \{0,1\}$, for the edges are introduced. The edges indicate that the two detections they connect are adjacent connections of the same track. One of the features used to form the edge weights will be the temporal difference of the detections, which allows for a penalty for missing detections to be learnt. Finally, $\hat{f}_k, \hat{l}_k \in \{0,1\}$ are introduced to indicate that v_k is the first, $\hat{f}_k$, and/or the last, $\hat{l}_k$, detection of a track. By denoting the combination of these indicators $x = (v_1, f_1, l_1, e_{1,2}, v_2, \cdots)$, the score, $f_{\text{score}}(x) =$

$$\sum_k \hat{f}_k s_{\text{entry}} + \sum_k \hat{v}_k f_{\text{detect}}\left(v_k\right) + \\ + \sum_{k_1,k_2} \hat{e}_{k_1,k_2} f_{\text{edge}}\left(P_{k_1,k_2}, C_{k_1,k_2}, v_{k_1}, v_{k_2}\right) \tag{6}$$

can be optimized to find the best solution to the tracking problem. Here, s_{entry} is a negative number efficiently becoming a threshold on the total track score for a track not to be considered noise.

Constraints have to be introduced to ensure that it is a proper solution in the sense that each detection only belongs to a single track and that unconnected detections are considered false positives. These are the flow constraints with flow variables both on edges and on vertices [9]. It ensures that the outgoing flow of each vertex is the same as the incoming flow and equal to the flow variable of the vertex. The constraints are

$$\hat{v}_k = \hat{f}_k + \sum_{k_1} \hat{e}_{k_1,k} = \hat{l}_k + \sum_{k_2} \hat{e}_{k,k_2}. \tag{7}$$

The solutions, x, that fulfills this equation are considered feasible solutions and the set containing all of them is denoted S, which allows the tracking problem to be expressed as

$$\underset{x \in S}{\operatorname{argmax}} \, f_{\text{score}}\left(x\right). \tag{8}$$

2.4. Optimization

The multi target tracking problem can be formulated as the maximisation in Equation 8. It can be solved using a linear program. This is guaranteed to result in a integer solution as it exhibits the total unimodularity property [4]. A more efficient way is to convert the linear program into a classical network cost flow problem [25] by replacing each vertex with two vertexes connected with a single edge with the original vertex weight as the edge weight and placing all incoming edges on one of these vertexes and all outgoing edges on the other. This network flow problem can then be solved using Bellmann-Ford [5] or, more efficiently, using Successive Shortest Paths [1]. Yet another alternative is to use K-shortest paths [4]. There are also solutions that can be used for online tracking [12].

3. Parameter learning

The tracking model in the previous section contains some functions that needs to be learned from annotated training sequences. These sequences are training examples consisting of short videos fully annotated with multi object tracking ground truth. This training can be performed efficiently using Generalized Graph Differences[2].

Fully connected neural networks will be used as basic blocks to construct these functions. These blocks are parameterised with two parameters only, the number of layers and the number of features. All layers have the same number of features, see Figure 2.

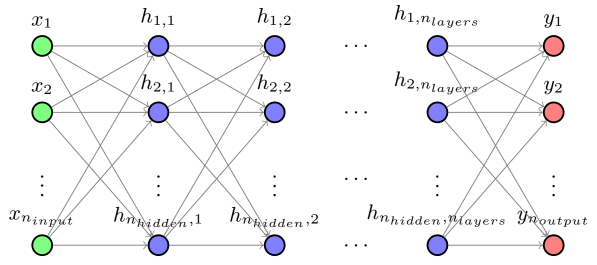


Figure 2. Generic fully connect neural network used in framework.

3.1. Model architecture

The parameters that needs to be learned are the scalar s_{entry} and the embedded parameters in the functions $f_{\text{detect}}\left(v_k\right)$ and $f_{\text{edge}}\left(P_{k_1,k_2}, C_{k_1,k_2}, v_{k_1}, v_{k_2}\right)$. These functions will be implemented as neural networks and it is the parameters of those networks that needs to learned together with s_{entry}.

The detection score, $f_{\text{detect}}\left(v_k\right)$, is a scalar valued function that depend on features extracted from the detection, v_k. The features used are

- The detection confidence, c_k.
- The maximum IoU between the detection v_k and any other detection in the same frame.
- The maximum IoA (intersection over area of v_k) between the detection v_k and any other detection in the same frame.

The detection score function, f_{detect}, will be realized as a small neural network with three inputs and one output, the detection score. The network has $n_{\text{detlayers}}$ fully connected hidden layers with n_{detfeat} features each.

The edge score, $f_{\text{edge}}\left(P_{k_1,k_2}, C_{k_1,k_2}, v_{k_1}, v_{k_2}\right)$ is more complicated and an overview of it is shown in Figure 3. It depend on all KLT-track connections, P_{k_1,k_2}, and all long connections, C_{k_1,k_2}, connecting the detections v_{k_1} and v_{k_2}, see example in Figure 4. The number of such connections will vary from edge to edge, as will the number of positions in the KLT-tracks. To handle that each KLT-track, $P_i \in P_{k_1,k_2}$ is converted into a fixed length feature vector, x_i^{KLT}, consisting of the features

- Temporal distance, $t_{k_2} - t_{k_1}$.
- Minimum confidence, $\min_j c_{i,j}$.
- The intersection over union between v_{k_2} and v_{k_1} translated according to the motion of the KLT-track P_i.
- A normalized trajectory shape consisting of P_i translated to place p_{i,j_1} (for $t_{i,j_1} = t_{k_1}$) at origin and then linearly interpolated into n_{linpkt} points placed uniformly spaced between t_{i,j_1} and t_{i,j_2}.

These feature vectors are processed, one by one, by a neural network, $f_{\text{KLT}}\left(x_i^{\text{KLT}}\right)$, with $n_{\text{kltlayers}}$ fully connected layers with n_{kltfeat} features each. That produces one feature vector for each KLT-track. They are then combined using average-pooling to form a single fixed length feature vector,

$$x_{k_1,k_2}^{\text{KLT}} = \frac{1}{\left|P_{k_1,k_2}\right|} \sum_{i \middle| P_i \in P_{k_1,k_2}} f_{\text{KLT}}\left(x_i^{\text{KLT}}\right). \tag{9}$$

This allows the varying number of KLT-tracks to be processes by a network construction with a fixed number of parameters and produce a feature vector of fixed length. Training this construction is similar to training a normal neural network while varying the batch size.

In a similar fashion, the long connections, $w_{i,k_1}^{\text{pre}} \in C_{k_1,k_2}$ are converted to fixed length feature vectors, x_i^{long}, with the features

- Temporal distance, $t_{k_2} - t_{k_1}$.
- The intersection over union between v_{k_2} and v_{k_1} translated according to the predicted velocity, $w_{i,k}^{\text{pre}}$.
- The predicted velocity, w_{i,k_1}^{pre}.
- The median post velocity of v_{k_2}, $\operatorname{median}_i\left(w_{i,k_2}^{\text{post}}\right)$.

These feature vectors are processed, one by one, by a neural network, $f_{\text{long}}\left(x_i^{\text{long}}\right)$, with $n_{\text{longlayers}}$ fully connected layers with n_{longfeat} features each, and averaged

$$x_{k_1,k_2}^{\text{long}} = \frac{1}{\left|C_{k_1,k_2}\right|} \sum_{i \middle| w_{i,k_1}^{\text{pre}} \in C_{k_1,k_2}} f_{\text{long}}\left(x_i^{\text{long}}\right). \tag{10}$$

The feature vectors, x_{k_1,k_2}^{klt} and $x_{k_1,k_2}^{\text{long}}$ are then concatenated and extended with the number of KLT-tracks, $\left|P_{k_1,k_2}\right|$ and the number of long connections, $\left|C_{k_1,k_2}\right|$ and passed to a final neural network, $f_{\text{combine}}\left(x_{k_1,k_2}^{\text{klt}}, x_{k_1,k_2}^{\text{long}}, \left|P_{k_1,k_2}\right|, \left|C_{k_1,k_2}\right|\right)$. This network has $n_{\text{combinelayers}}$ fully connected hidden layers with $n_{\text{combinefeat}}$ features each, and a single output, the edge feature $f_{\text{edge}}\left(P_{k_1,k_2}, C_{k_1,k_2}, v_{k_1}, v_{k_2}\right)$.

3.2. Loss function

The multi target tracking optimization problem, Equation 8, can, in theory, be solved by enumerating all feasible solutions, $x \in S$ (Equation 7), and picking the one that maximizes $f_{\text{score}}\left(x\right)$ (Equation 6). The score function, $f_{\text{score}}\left(x\right)$, is a linear combination of the outputs of several invocations of the neural networks defined above. That means the entire function, $f_{\text{score}}\left(x\right)$, is differentiable and can be learned using for example SGD.

The function can be seen as an embedding that embeds feasible solutions, x, into a one-dimensional feature space with the one property that better solution should have higher score.

The embedding can be learnt from training data consisting of ordered pairs of feasible solutions, (x^*, x) where x^* is the correct globally optimal solution and x is any other feasible solution. Details of how x is created in practice will

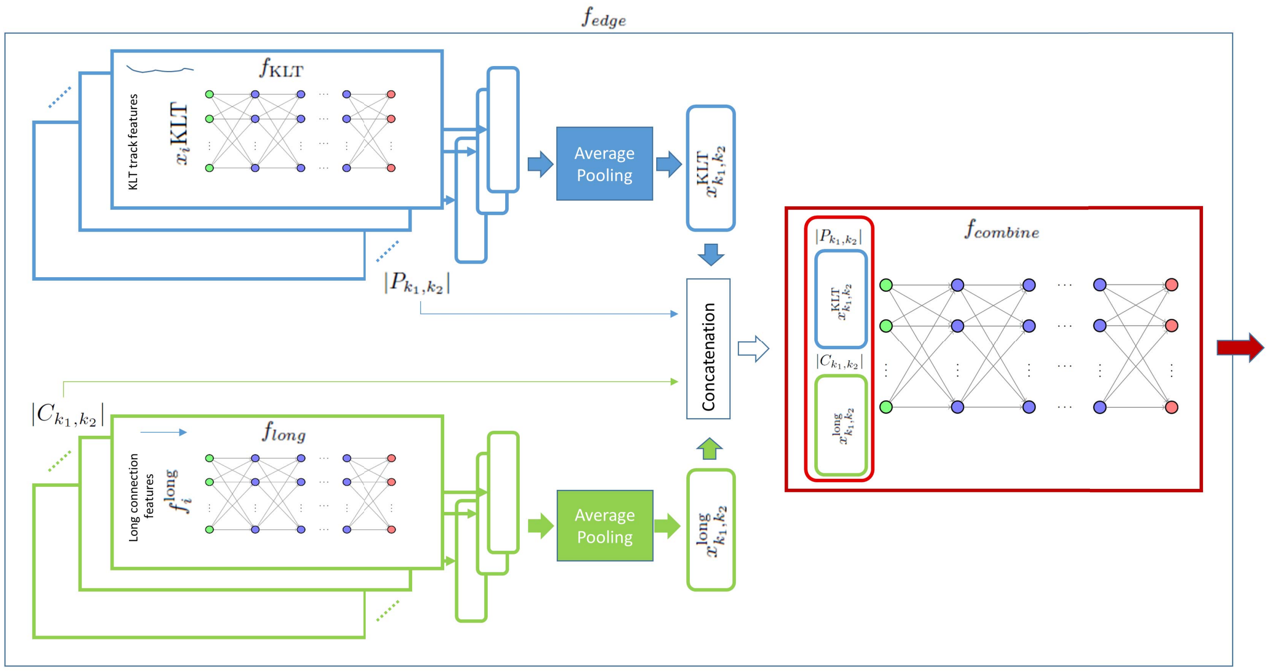


Figure 3. The architecture of $f_{\text{edge}}\left(P_{k_1,k_2}, C_{k_1,k_2}, v_{k_1}, v_{k_2}\right)$.

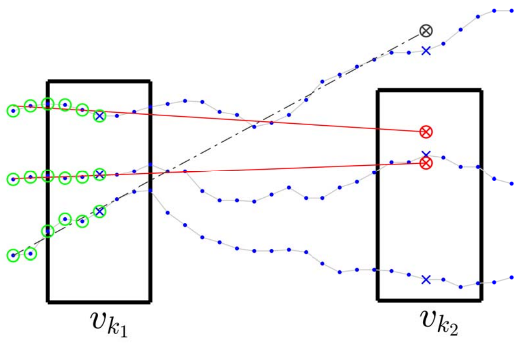


Figure 4. A pair of detections, (v_{k_1}, v_{k_2}) (black boxes) connected with $\left|P_{v_{k_1},v_{k_2}}\right| = 2$ KLT-tracks (blue dots) and $\left|C_{v_{k_1},v_{k_2}}\right| = 2$ long connections (red lines) estimated from $n_{\text{velest}} = 6$ positions (green circles).

be discussed later, but in general consider it to be a modification of x^*. The learning is achieved using the ranking loss

$$-\log\sigma\left(f_{\text{score}}\left(x^*\right) - f_{\text{score}}\left(x\right)\right), \tag{11}$$

where σ is the sigmoid function. It is used here to suppress large differences as the only property from the pair of interest is that $f_{\text{score}}\left(x^*\right)$ should be larger than $f_{\text{score}}\left(x\right)$. Note that it is the ranking of these that are important, and how much larger one is over the other is not that interesting, since the following linear program will find the best one. Another way to motivate this loss is to derive it from a binary classifier trained using a sigmoid activation and cross entropy loss to produce whether x^* or x is the correct solution. Such a detector would be trained using both positive and negative samples. The loss for a positive sample, (x^*, x), is in that case the same as in Equation 11. For a negative sample, (x, x^*), the loss is

$$-\log\left(1 - \sigma\left(f_{\text{score}}\left(x\right) - f_{\text{score}}\left(x^*\right)\right)\right), \tag{12}$$

which is also equal to Equation 11 since $1 - \sigma\left(-x\right) = \sigma\left(x\right)$.

3.3. Generalized Graph Differences (GGD)

The feasible solutions, x^* and x, can be represented using graphs constructed from the tracking graph by only keeping the edges and vertices with positive flow, i.e. the positive elements of x^* and x. All terms common to both x^* and x will cancel each other out in the difference in Equation 11. This means that only the terms that differ needs to be considered. That is a concept, introduced in [2], called a Generalized Graph Difference, or GGD. It consists of

- A set of edges consisting of the edges in x^* but not in x with the same weights and the edges in x but not in x^* with negated weights.

- A set of vertices consisting of the vertices in x^* but not in x with the same weights and the vertices in x but not in x^* with negated weights.

Note that a generalized graph difference is typically not a graph, or even a generalized graph, as it can contain edges not connected to any vertices. Thus, the focus here is on the graph-difference in a general sense when referring to general graph difference.

This is interesting because hard examples consists of cases where x^* and x are very similar and thus have a lot of terms in common, resulting in small generalized graph differences. Such differences can be constructed by looking at annotated sequences where the optimal solution is known and introducing small errors by changing one or a few edges to form another feasible solution. Several such modifications are introduced in [2] representing common errors made by trackers. These are shown in Table 3, 4, 5 and 6.

Applying these modifications to every position where they apply in a ground truth graph results in a lot of GGDs that can be generated and used for training.

Each generated training example is constructed by taking a single ground truth graph and applying a single modification to a single position. Hence each batch will contain one mistake per example. All possible such examples are generated and will form one epoch.

In addition to the GGDs introduced by the modifications suggested in [2], two new ways of generating GGDs were also used:

LongConnectionOrder Each detection will have several edges to future detections belonging to the same track at different future frames. These are all correct connections in the sense that they connect detections of the same object. However the connection skipping one or several frames should be interpreted as a claim that the object were not detected in the skipped frames. This means that solutions skipping frames with true positive detections should be assigned a lower score than solutions including those detections even though both solutions are in some sense correct. Since the loss function used is a ranking loss, this is possible without having to consider any of the solution a negative example. To encourage this, the outgoing connections of each detection going to another detection of the same object is order by the number of frames they skip. Then a GGD is formed for each adjacent pair of such connections with the shorter of the two augmented with ground truth connections until it reaches the same detection as the longer connection.

LongFalsePositiveTrack Long false positive tracks can be formed by starting at any false positive detection that have no incoming connections from other false positive detections and form a track by recursively adding the next connected false positive detection that skips the least amount of frames. Such tracks should score less than the empty solutions which gives another form of GGD.

	MOTA	FN	FP	Frag.	IDF1
GT	43.3	41068	112	1981	60.6
GT Interpolated	48.4	32802	4686	422	67.9
Proposed	23.4	42917	12630	1314	48.2
Interpolated	22.6	41802	14366	893	48.6

Table 1. Results on VisDrone2019 validation set. GT lines uses the ground truth to assign ID's to detections and gives an upper bound to the scores for trackers that simply connects the provided detections. Proposed and Interpolated are the proposed approach where the holes in the tracks have been filled using linear interpolation in the later.

This will produce several additional general graph differences which are added to the training data used to train the tracker.

4. Experiments

The presented tracker were trained on the VisDrone2019 dataset [26] and evaluated on the validation set and test set. The provided public FasterRCNN [19] detections were used. Training details and hyperparameters were identical to the original tracking paper [2]. Except for the n_{minlen}, which were set to 20.

The results for the validation set are shown in Table 1 and compared to an optimal solution that uses the ground truth to assign track ID's to detections. This gives an upper bound to the scores for a tracker that simply connects the provided detections. Note that this optimal tracker do make 112 false positive detections (top row in Table 1). This is due to annotation noise causing the class label of one of the ground truth tracks to varies between car and van from frame to frame. In Table 2 the Average Precision (AP) overall and for each category can be found for the test set of the submitted Generalized Graph Difference Tracker (GGDTRACK) at VisDrone 2019 together with baselines provided by the VisDrone Team[22]. Some scores can be improved by filling in the holes in the produced tracks using linear interpolation, but this will also increase the false positive detections which will for example worsen the MOTA score.

The tracker performs fairly well. Especially in the IDF1 score. The MOTA score is however not as good. The main reason for that is a high count of false negative detections. But 95.7% (or 78.4% in the interpolated case) of those failures are also present in the optimal solution, which means the main problem here is the detector and not the presented tracking framework.

There is also a fair amount of false positive detections (17%, or 19% if interpolated). Some of these can be explanted by the fact that the detector does not always agree with the annotators on exactly which frame each object en-

	AP	AP@0.25	AP@0.50	AP@0.75	AP car	AP bus	AP truck	AP ped	AP van
Proposed	23.09	31.01	22.7	15.55	35.45	28.57	11.9	17.2	22.34
Ctrack [27]	16.12	22.40	16.26	9.70	27.74	28.45	8.15	7.95	8.31
CMOT [3]	14.22	22.11	14.58	5.98	27.72	17.95	7.79	9.95	7.71
GOG [17]	6.16	11.03	5.30	2.14	17.05	1.80	5.67	3.70	2.55
TBD [10]	5.92	10.77	5.00	1.99	12.75	6.55	5.90	2.62	1.79
CEM [16]	5.70	9.22	4.89	2.99	6.51	10.58	8.33	0.70	2.38
H2T [21]	4.93	8.93	4.73	1.12	12.90	5.99	2.27	2.18	1.29
IHTLS [8]	4.72	8.60	4.34	1.22	12.07	2.38	5.82	1.94	1.40

Table 2. Results on VisDrone2019 test set for the Generalized Graph Differences tracker compared with baselines provided by the VisDrone Team[22].

ters or leaves the scene. But even if they were somehow ignored, it is likely that a tendency for connecting false positive detections into tracks rather than discarding them is the main weakness of the presented algorithm.

5. Conclusions

We have presented a method that can learn the weights of a network flow tracker from Generalized Graph Differences. That is an efficient representation of differences between graphs. Two modifications from the original work [2] has been introduced, see details in section 3.3. The algorithm were evaluated in the VisDrone2019 competition, where it outperforms the baselines and are ranked 6th among all the teams. Training data were produced from small perturbation of ground truth tracks which allows the model to be trained using the standard Adam optimizer. There is no need to solve an additional optimisation problem for each example in each training batch as some prior work do. The method resulted in an overall average precision of 23.09 in the test set of VisDrone 2019.

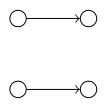
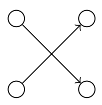
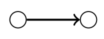
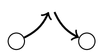
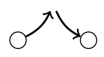
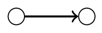
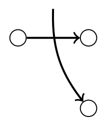
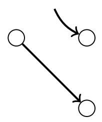
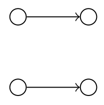
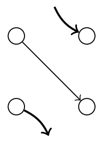
Name	Ground Truth	Possible Error
ID Switch		
Split		
Merge		
Split and Merge		
Double Split and Merge		

Table 3. Switch, split and merge errors introduced to form training data pairs from ground truth.

6. Acknowledgement

The computations were performed on resources provided by the Swedish National Infrastructure for Computing (SNIC) at LUNARC.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[2] H. Ardö and M. Nilsson. Multi target tracking by learning from generalized graph differences. *CoRR*, abs/1908.06646, 2019.

[3] S.-H. Bae and K.-J. Yoon. Robust online multi-object tracking based on tracklet confidence and online discriminative appearance learning. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 1218–1225, Washington, DC, USA, 2014. IEEE Computer Society.

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

[5] D. Bertsekas and R. Gallager. *Data Networks (2Nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.

[6] Z. Cao, T. Simon, S. Wei, and Y. Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1302–1310, July 2017.

[7] J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, edi-

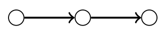
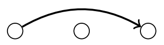
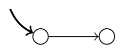
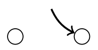
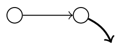
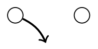
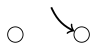
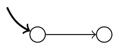
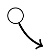
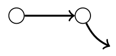
Name	Ground Truth	Possible Error
Detection Skip		
Skip First		
Skip Last		
Extra First		
Extra Last		

Table 4. Skip and extra errors introduced to form training data pairs from ground truth.

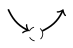
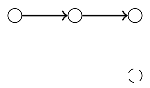
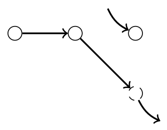
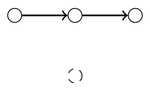
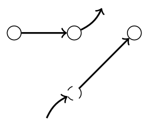
Name	Ground Truth	Possible Error
False positive		
Split to False Positive		
Split from False Positive		

Table 5. False positives introduced to form training data pairs from ground truth.

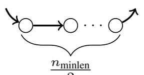
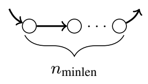
Name	Ground Truth	Possible Error
Too Short Track		
Proper Track		

Table 6. Track lengths data pairs used for training.

tors, *Advances in Neural Information Processing Systems 29*, pages 379–387. Curran Associates, Inc., 2016.

[8] C. Dicle, O. I. Camps, and M. Sznaier. The way they move: Tracking multiple targets with similar appearance. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2013.

[9] D. Frossard and R. Urtasun. End-to-end learning of multi-sensor 3d tracking by detection. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 635–642, May 2018.

[10] A. Geiger, M. Lauer, C. Wojek, C. Stiller, and R. Urtasun. 3d traffic scene understanding from movable platforms. *IEEE transactions on pattern analysis and machine intelligence*, 36, 09 2013.

[11] C. Kim, F. Li, A. Ciptadi, and J. M. Rehg. Multiple hypothesis tracking revisited. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4696–4704, Dec 2015.

[12] P. Lenz, A. Geiger, and R. Urtasun. Followme: Efficient online min-cost flow tracking with bounded memory and computation. In *ICCV 2015*, pages 4364–4372, 12 2015.

[13] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, July 2017.

[14] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, and A. Berg. Ssd: Single shot multibox detector. In B. Leibe, J. Matas, M. Welling, and N. Sebe, editors, *Computer Vision - 14th European Conference, ECCV 2016, Proceedings*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pages 21–37, Germany, 1 2016. Springer Verlag.

[15] W. Luo, X. Zhao, and T. Kim. Multiple object tracking: A review. *CoRR*, abs/1409.7618, 2014.

[16] A. Milan, S. Roth, and K. Schindler. Continuous energy minimization for multitarget tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(1):58–72, Jan. 2014.

[17] H. Pirsiavash, D. Ramanan, and C. C. Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 1201–1208, Washington, DC, USA, 2011. IEEE Computer Society.

[18] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, July 2017.

[19] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015.

[20] E. Ristani, F. Solera, R. S. Zou, R. Cucchiara, and C. Tomasi. Performance measures and a data set for multi-target, multi-camera tracking. In *Computer Vision - ECCV 2016 Workshops - Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part II*, pages 17–35, 2016.

[21] L. Wen, W. Li, J. Yan, Z. Lei, D. Yi, and S. Z. Li. Multiple target tracking based on undirected hierarchical relation hypergraph. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 1282–1289, Washington, DC, USA, 2014. IEEE Computer Society.

[22] L. Wen, P. Zhu, et al. Visdrone-mot2019: The vision meets drone multiple object tracking challenge results. In *ICCV Workshops*, 2019.

[23] B. Xiao, H. Wu, and Y. Wei. Simple baselines for human pose estimation and tracking. In *European Conference on Computer Vision (ECCV)*, 2018.

[24] J. yves Bouguet. Pyramidal implementation of the lucas kanade feature tracker. *Intel Corporation, Microprocessor Research Labs*, 2000.

[25] L. Zhang, Y. Li, and R. Nevatia. Global data association for multi-object tracking using network flows. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*, 2008.

[26] P. Zhu, L. Wen, X. Bian, H. Ling, and Q. Hu. Vision meets drones: A challenge. *CoRR*, abs/1804.07437, 2018.

[27] P. Zhu, L. Wen, et al. In *The European Conference on Computer Vision (ECCV) Workshops*, September 2018.