FollowMe: Efficient Online Min-Cost Flow Tracking with Bounded Memory and Computation

Philip Lenz
Karlsruhe Institute of Technology
lenz@kit.edu

Andreas Geiger
MPI Tübingen
andreas.geiger@tue.mpg.de

Raquel Urtasun
University of Toronto
urtasun@cs.toronto.edu

Abstract

One of the most popular approaches to multi-target tracking is tracking-by-detection. Current min-cost flow algorithms which solve the data association problem optimally have three main drawbacks: they are computationally expensive, they assume that the whole video is given as a batch and they scale badly in memory and computation with the length of the video sequence. In this paper, we address each of these issues. Our first contribution is a dynamic version of the successive shortest-path algorithm which solves the data association problem optimally while reusing computation, resulting in one order of magnitude faster inference than standard solvers. With our second contribution we address the optimal solution to the data association problem when dealing with an incoming stream of data (i.e., online setting). Our last contribution is an approximate online solution with bounded memory and computation, which can handle videos of arbitrarily length while performing tracking in real time. We demonstrate the effectiveness of our algorithms on the KITTI and PETS2009 benchmarks and show state-of-the-art performance, while being significantly faster than existing solvers.

1. Introduction

Despite decades of research, robust multi-target tracking (i.e., following a variable and unknown number of objects over time) remains one of the fundamental open problems in computer vision. Tracking algorithms have to tackle two main challenges: handling occlusion and dealing with the variable and unknown number of objects. The former leads to association errors, fragmentations and identity switches, while the latter requires inference algorithms to solve the difficult problem of model selection.

Tracking-by-detection has proven as one of the most successful strategies in recent years [40, 31, 8, 10, 26, 37, 29, 11, 32]. Objects are detected independently in each frame, followed by an association step linking individual detections to form trajectories. The association problem is difficult due to the presence of occlusions, noisy detections as well as false negatives. Early attempts establish associations between pairs of consecutive frames, a problem which can be solved optimally in polynomial time using the Hungarian method. More recently, sophisticated discrete-continuous optimization schemes have been proposed [3], which alternate between optimizing trajectories and performing data association over the whole sequence by assigning detections to the set of trajectory hypotheses. Unfortunately, the resulting optimization problems are highly non-convex and provide no guarantees of optimality.

In their seminal work, Zhang et al. [40] showed how multi-frame data association can be cast as a network flow problem. The optimal solution is found by a min-cost flow algorithm, solving simultaneously for the number of objects and their trajectories according to a cost-function which incorporates the likelihood of a detection as well as pairwise relationships between detections in two consecutive frames. Recent extensions [11] show how higher-order motion models can be incorporated into the formulation at the cost of giving up on global optimality.

In this paper, we address three fundamental challenges of min-cost flow data association: First, the high computational complexity of existing optimal algorithms prohibits their application to large video sequences. To tackle this problem, we propose a dynamic solver based on the distributed Bellman-Ford algorithm for package routing, which leverages the special structure of tracking networks and reuses computation while computing consecutive shortest paths. The second limitation of current global data association algorithms is that they require a batch setting, i.e., they can only be applied to the full video sequence. This is a non-realistic assumption for many tracking scenarios. To
address this, we propose an online algorithm, which integrates novel frames into the current solution in an optimal fashion and yields additional speed ups over the proposed dynamic solver. The last fundamental challenge we address is the unbounded growth of existing min-cost flow solvers in terms of memory and computation. This prevents their implementation in real-world applications such as autonomous driving or surveillance, where tracking algorithms are required to run continuously for hours or even weeks. Towards this goal, we exploit the fact that the scope of object trajectories is often limited in time and derive an online algorithm which is able to estimate near-optimal trajectories in only a fraction of the time required by standard solvers while at the same time being bounded in memory and computation. This allows us to solve videos of arbitrary length in real-time.

We demonstrate the effectiveness of the proposed solvers on the KITTI [19] and PETS 2009 [15] tracking benchmarks. As evidenced by our experiments, the proposed methods significantly outperform existing algorithms in terms of the number of relaxations required to solve the problem and result in state-of-the-art performance. Our code will be made available upon publication.

2. Related Work

Multi-target tracking approaches can be divided into two main categories: filtering-based approaches and batch methods. Filtering-based methods [30, 10, 16] are based on the Markov assumption, i.e., the current state depends only on the previous state. While they are fast and applicable in real-time applications, they typically suffer from their inability to recover from early errors.

Recent efforts focus primarily on batch methods, where object hypothesis are typically obtained using an object detector (tracking-by-detection) and tracking is formulated as an optimization problem over the whole sequence. This mitigates the aforementioned problems and allows for the integration of higher-level cues such as group behavior in the case of pedestrian tracking [13, 23, 21]. In [2], the task is formulated as a continuous energy minimization problem which allows incorporating richer motion models but is difficult to optimize. Discrete-continuous optimization techniques have been proposed [8, 29], where discrete (data association) and continuous (trajectory fitting) optimization are iterated until converging to a hopefully better local minimum. In [5, 9, 12], approximate Markov chain Monte Carlo techniques are employed for solving the data association problem. In order to increase the discriminative power of appearance and dynamical models, online learning approaches have been suggested [25, 27, 30, 33, 37, 38, 39, 41]. In [7, 21], the problem is cast as optimization on a grid using a linear program while assuming that the observing camera is static. The problem of tracking through occlusions has been tackled in [22, 35, 34, 28] by using context from outside the object region and by building strong statistical motion models.

While all of the aforementioned formulations resort to approximate optimization without optimality guarantees, Zhang et al. [40] showed how data association with pairwise energies can be formulated as a network flow problem to which standard graph solvers can be used to retrieve the global optimum. Their formulation solves for the globally optimal trajectories including their number, and hence implicitly solves the model selection problem. [11, 4] consider including higher-order terms into the problem formulation at the price of loosing optimality. To reduce the computational complexity of min-cost flow algorithms, Pirsiavash et al. [31] proposed to use the successive shortest-path algorithm and devised an efficient greedy approximation scheme.

In contrast to the aforementioned works, in this paper we propose an efficient, globally optimal online algorithm, which is able to process frame-by-frame while reusing computation via efficient caching strategies. We start by devising a variant of the distributed Bellman-Ford algorithm (“oDBF”) in Section 3, which is tailored to the properties of the network flow formulation for multi-target tracking. Section 3 extends this approach to the online setting by allowing frames to be added to the current solution. Our first online algorithm (“oDBF”) yields the optimal solution by considering all frames up to the current frame (Section 4.1).

To consider sequences of arbitrary length, we further present a memory-bounded version of the online algorithm in Section 4.2 which, despite violating optimality, delivers high-quality object tracks maintaining track ids over long periods. Section 5 provides our experimental evaluation where we compare the proposed algorithms against state-of-the-art [9, 10, 31, 18] on the challenging KITTI [19] and PETS2009 [15] datasets.

3. Optimal Tracking via Dynamic Computation

One of the most popular approaches to multi-target tracking is tracking-by-detection, where a set of detections are computed for each frame and trajectories are formed by linking the detections. In this section, we start by reviewing how to formulate the multi-target tracking problem as a min-cost flow problem. We then propose a dynamic algorithm which exploits the distributed Bellman-Ford strategy, and is able to compute optimal trajectories in a fraction of the time of
standard solvers. In Section 3.4 we extend this dynamic algorithm to the online setting, i.e., when dealing with an incoming stream of frames instead of the common batch setup, and show that this can decrease runtime even further.

3.1. Tracking as Min-Cost Flow

Following tracking-by-detection approaches, we assume that a set of detections $\mathcal{X} = \{x_i\}$ is available as input. Let $x_i = (x_i, t_i, s_i, a_i, d_i)$ denote a detection, with $x_i$ the position of the bounding box, $t_i$ the time step (frame index), $s_i$ the size of the bounding box, $a_i$ the appearance, and $d_i$ the detector score. We define a trajectory as an ordered set of observations $T_k = \{x_{k_1}, x_{k_2}, \ldots, x_{k_l}\}$. An association hypothesis is then defined as a set of trajectories $\mathcal{T} = \{T_k\}$.

The data association problem can be formulated as maximizing the posterior probability of $\mathcal{T}$ given the observations $\mathcal{X}$, subject to the constraints that trajectories should not overlap, i.e., $T_k \cap T_l = \emptyset$, $\forall k \neq l$. Assuming independent observations and trajectories, respectively, we obtain the optimal set of trajectories $\mathcal{T}^*$ as:

$$\mathcal{T}^* = \arg\max_{\mathcal{T}} \prod_i P(x_i | \mathcal{T}) \prod_{T_k \in \mathcal{T}} P(T_k) \quad (1)$$

We further assume a first order Markov model, i.e., links are only defined between detections in two consecutive frames. Note that this can be easily generalized by adding additional edges in the network. Given a set of trajectories $\mathcal{T}$, the task can be formulated as an energy minimization problem

$$\mathcal{T}^* = \arg\min_{\mathcal{T}} \sum_i C_{en,i}f_{en,i} + \sum_{i,j} C_{i,j}f_{i,j} \quad (2)$$

$$+ \sum_i C_{ex,i}f_{ex,i} + \sum_i C_i f_i$$

$$s.t. \ f_{en,i} + \sum_j f_{j,i} = f_i = f_{ex,i} + \sum_j f_{i,j} \ \forall i \quad (3)$$

where $C_{en,i}$ is the cost of creating a new trajectory at $x_i$ and $C_{ex,i}$ is the cost of exiting a trajectory at $x_i$. The cost of linking two consecutive detections $x_i$ and $x_j$ is denoted $C_{i,j}$ and $C_i$, encoding the cost of $x_i$ being a true detection or a false positive (data term). Furthermore $f$ is a binary flow variable, taking 1 if the edge is part of a trajectory and 0 otherwise. The constraints ensure that detections can only be associated with at most one trajectory.

In their seminal work [40], Zhang and Nevatia showed how to map this problem into a min-cost flow network $G(\mathcal{X})$. This is illustrated in Fig. 4 where for each observation two nodes $u_i$ and $v_i$ are created (summarized as one node for clarity of illustration) with an edge between them with cost $c(u_i, v_i) = C_i$ and flow $f(u_i, v_i) = f_i$. For entering and exiting trajectories, edges between the source $s$ and the first variable (with cost $c(s, u_i) = C_{en,i}$ and flow $f_{en,i}$), and the second variable and the sink $t$ (with cost $c(v_i, t) = C_{ex,i}$ and flow $f_{ex,i}$) are introduced. Finally, edges between consecutive detections $(v_i, u_j)$ encode pairwise association probabilities with cost $c(v_i, u_j) = C_{i,j}$ and flow $f_{i,j}$.

To find the optimal solution, Zhang et al. [40] start with flow zero, and augment the flow one unit at a time while applying the push relabeling algorithm [20] to retrieve the shortest path at each iteration. The algorithm terminates if the cost of the currently retrieved shortest path is greater or equal to zero. For efficiency, the bisection method can be applied on the number of flow units, reducing time complexity from linear to logarithmic with respect to the number of trajectories. The total complexity is then $O(mn^2 \log n)$, with $n$ number of nodes and $m$ number of edges. Recently, [31] proposed to use the successive shortest-path algorithm (“SSP”, Algorithm 3 [31]) for computing the optimal set of trajectories, which reduces computational complexity to $O(KN^2)$ where $K$ is the number of trajectories and $N$ denotes the number of frames.
3.2. Successive Shortest-Path (SSP) for Min Cost Flow

We start our discussion by describing the Bellman-Ford algorithm (“BF”, Algorithm 2), which is typically called at each iteration of the successive shortest-path algorithm (“SSP”) for computing the shortest path on the residual graph. BF is based on the principle of relaxation, in which an upper bound of the correct distance is gradually replaced by tighter bounds (by computing the predecessor and its distance) until the optimal solution is reached. To achieve optimality the BF algorithm relaxes all edges in the graph for \( n - 1 \) iterations, where \( n \) denotes the number of vertices in the graph.

While the BF algorithm is able to compute a single trajectory with the lowest cost, in multi-target tracking we are interested in recovering the optimal set of trajectories. In the following, we review how to compute the optimal set of trajectories by means of SSP (see Algorithm 1) for a given example. Let us consider the directed acyclic graph (“DAG”) in Fig. 2 (a) which specifies an instance of the network flow problem formulated in Eq. 3 for four frames and three detections per frame. Note that in general we do not require the number of detections to be the same for each time step. For clarity, we only sketch edges coming from the source and going to the target. First, we relax all edges by traversing the graph from left-to-right, which is illustrated in Fig. 2 (b+c) for the first and second time step, respectively. This allows to recover the first optimal trajectory depicted in green in Fig. 2 (d). We refer to this algorithm as “DAG-SP” as stated in [14] p. 655. An alternative at this point is to remove all edges from this trajectory and run the algorithm again [31]. However, this greedy version of the algorithm (called “DP” in the following) does not guarantee optimality, as shown in Fig. 2 (e) where some of the edges are absent in the graph. Instead, the SSP algorithm computes a residual graph (Fig. 2 (f)) by inverting the edges of the previously found optimal trajectory. As the resulting graph is not a DAG anymore, BF is applied again (Fig. 2 (g)) yielding the second shortest path. The process finishes when a newly found shortest path has positive costs, i.e., it can not further reduce the total cost specified in Eq. 3. The final trajectories shown in Fig. 2 (h) are recovered by extracting all backward edges from the most recent residual graph, starting the back-tracking procedure at the target node.

3.3. Dynamic Min-Cost Flow (DBF)

The key component at every iteration of the SSP algorithm is the broadcasting strategy used to update the predecessors in the residual graph. Using the standard Bellman-Ford algorithm for this purpose will lead to }^2 \text{ messages for each trajectory, which is computationally very expensive. Instead, in this section we propose a variant of the distributed Bellman-Ford algorithm (called “DBF” in the following) tailored to multi-target tracking problems, which reuses computation, only updating predecessors when needed. This information is available to us, as at each point in time we are aware of the changes we have applied to the graph for computing the new residual graph. We take advantage of this information by adopting a depth-first search (“DFS”, [14] p.603) strategy which detects all subsequent nodes with invalid predecessors. While traversing the graph in temporal direction, we update outdated predecessors using information about nodes in the previous time steps. Nodes which have been processed in this forward
Figure 2. **Illustration of the SSP Algorithm.** The original problem is depicted in (a). The BF algorithm relaxes the red edges in (b) and (c) yielding the shortest path in the DAG, depicted in green in (d). Simply removing this path and re-running BF violates optimality as shown in (e). Instead, a SSP strategy on the residual graph (f) allows for finding the optimal solution (g+h).

Figure 3. **Dynamic Message Broadcasting:** (a) For a given residual graph (the shortest path marked in blue), nodes with invalid predecessors are detected (in red). For each node, the current best predecessor from the previous time step is selected among all candidates (connected by red edges). Nodes from (a) are put into a queue. (b) The queue is maintained, taking successively nodes from the queue and relax their backward pointing edges, starting at the target node. Nodes with updated predecessors (in red) are put into the queue. (c) The queue is maintained in forward direction as it was in the last step. (b) and (c) are iterated until no nodes are left in the queue, which leads to the final solution for this particular residual graph (d).

We illustrate our DBF algorithm in Fig. 3 for the example from Fig. 2. Given the trajectory found in Fig. 2(d), we revert its edges to form the residual graph of Fig. 2(a). Note that the corresponding predecessor mappings have to be updated as the direction of these edges has changed. We start by updating all predecessors for nodes belonging to the most recent trajectory (blue path in Fig. 3(a)) in a forward sweep (relaxed edges are marked in red). Next, all nodes with a new predecessor propagate their cost along their respective shortest path in a backward sweep (Fig. 3(b)). Finally, the same information is propagated in a forward sweep (c). These steps are iterated until costs do not change anymore. The algorithm terminates and a new shortest path is found (d).

4. **Online Solution**

We now extend the dynamic algorithm (DBF) presented in Section 3.3 to the online setting ("oDBF")
Algorithm 3: Data association by SSP using DBF

Input: Detections $X = \{x_t\}$

Output: Set of trajectory hypotheses $T = \{T_k\}$

1. $k = 0$ // shortest path counter
2. $G(V, E, C, f) \leftarrow \text{ConstructGraph}(X)$
3. $f(G) = 0$ // initialize flow to 0
4. $G_r(f) \leftarrow G(f)$ // initialize $G_r(f)$ as a DAG
5. $\gamma_0, \pi_0 \leftarrow \text{DAG-SP}(G_r(f))$
6. $q \leftarrow \emptyset$ // $q$ is maintained for every iteration $k$
7. while $C(\gamma_k) < 0$ do
8.   $k \leftarrow k + 1$ // find next shortest path
9.   $\pi_k \leftarrow \pi_{k-1}$ // predecessor map $\pi_k$ is initialized
10. $q \leftarrow \emptyset$ // predecessors for last path are invalid
11. wait $q \neq \emptyset$ do // ProcessDBFQueue
12.   foreach node $\in q$ do
13.      $q \leftarrow q \setminus$ node
14.      $\pi_k \leftarrow \text{Update}(\pi_k, \text{node})$
15.      if updated then
16.         $q \leftarrow \text{AddSuccessors}(q, \text{node}, G)$
17.   endforeach queue in time direction
18.   foreach node $\in q$ do
19.      $q \leftarrow q \setminus$ node
20.      $\pi_k \leftarrow \text{Update}(\pi_k, \text{node})$
21.      if updated then
22.         $q \leftarrow \text{AddSuccessors}(q, \text{node}, G)$
23.   endforeach queue against time direction
24. if $C(\gamma_k) < 0$ then
25.   // revert edges for $G_r(f)$ along $\gamma_k$
26.   $G_r(f) \leftarrow \text{Update}(G_r(f), \gamma_k)$
27. return $T$

leading to further reductions in runtime while maintaining optimality. We then present a memory-bounded algorithm ("mboDBF"), which can estimate trajectories in a principled way in large graphs, where existing algorithms fail due to their unbounded memory and computation requirements. Our memory bounded algorithm computes locally optimal trajectories for a time window of $\tau$ frames in a dynamic fashion. Previous trajectories, predecessor maps and residual graphs are held in a cache and updated at each iteration. By collapsing paths of nodes which leave the current time window, our algorithm is able to remember previously found trajectories and thus maintains track identities. While optimality can not be guaranteed anymore, our experiments show little loss in performance with respect to batch processing on the full sequence. The time window size $\tau$ controls the tightness of the approximation and for $\tau = N$ our mboDBF algorithm reduces to the oDBF algorithm which is optimal.

4.1. Optimal Online Solution (oDBF)

Fig. 4 illustrates the idea of our oDBF algorithm: Consider the network specified in (a) and its current solution in (b) for which the optimal set of trajectories and predecessor maps are available. As a new frame arrives (c), new nodes are added extending the graph to the next time step. The first trajectory from the previous time step is extended by running one step the DAG-SP solver, since all edges point forward. For the subsequent iterations of the successive shortest path algorithm, the predecessor maps from the previous time steps can be reused if the paths under consideration did not change their order. In this case, we use the predecessor map marked in magenta in Fig. 4(e). Note that the second trajectory in cyan has already been found up to the previous time step. The new nodes marked in orange are added to a queue, which is maintained by the following iterative steps. In this example, only the predecessor of the node marked in blue in Fig. 4(f) is updated. This node broadcasts a message to its successors (red edges). As no changes take place, the algorithm terminates. The final result for frame $t = 4$ is illustrated in Fig. 4(g).

Unfortunately, this simple caching strategy alone is often not sufficient in practice. In the worst case, a non-cached trajectory is found and the regular DBF algorithm must be run since the predecessor map from the previous time step is outdated. For the simple caching strategy described above, this happens primarily in the presence of competing trajectories with similar costs, frequently changing their ordering.

To tackle this problem, we extend our caching strategy in the following way. We keep all predecessor maps for a cache length of $\Delta$ frames in memory. If a trajectory from the previous time step is not extended, the cache is searched for a match. If the cache contains a match from an older time step $\delta_i$ (trajectories for this iteration are equal up to $t_{max} - \delta_i$), the cache is validated (having the same history in trajectory space as in the current iteration), used as initialization for the following iterative process, and all nodes starting at $t_{max} - \delta_i$ as well as all nodes with predecessors detected as invalid (previously found paths) are added to a queue. We process this queue using the previously described dynamic message broadcasting strategy for DBF which efficiently leads to the optimal solution.
Figure 4. **Online (a)-(g) and Memory Bounded (h) Algorithm:** Assume that for the original network at $t = 3$ (a), the solution (b) is already known. A new set of detections (blue nodes) for $t = 4$ arrives and is connected to the graph (c) resulting in a DAG for which the shortest path can be found by applying one step of the DAG-SP algorithm (d). The predecessor map from the cache for the results up to $t = 3$ (in magenta) is re-used and the orange nodes are added to the queue (e). The queue is maintained and new predecessors are found (f). After one iteration, the algorithm terminates and the optimal solution is found (g). For the mboDBF algorithm, paths which pass through nodes in the oldest time step are merged into the entry costs of the next time step (h).

### 4.2. Memory-Bounded Near Optimal Online Solution (mboDBF)

While the oDBF strategy proposed in the previous section is more efficient than simply running DBF for every time step, it still scales poorly to very large problems since messages might broadcast back to frame one in order to guarantee optimality. Furthermore, storing graphs of arbitrary length in memory is infeasible in practice. Thus, in this section we devise a memory and computationally bounded approximation which we call “mboDBF” and which is illustrated in Algorithm 4 and Fig. 4.

As this algorithm extends the previously described oDBF algorithm, we will now review the differences between oDBF and mboDBF. Consider an optimized time window of $\tau = 4$ frames where the solution from the previous time step is given by the graph in Fig. 4 (g). Before adding new nodes to the graph for the most current time step $t_{\text{max}}$, the oldest time step $t_{\text{max}} - \tau$ is removed from the graph as illustrated in Fig. 4 (h) to maintain a memory/computation budget. Simply deleting edges from the graph will be very suboptimal as this completely discards computations from the previous time steps. In order to “remember” known paths, we map the predecessors for nodes at $t_{\text{max}} - \tau$ to the respective entry edge and delete all remaining edges.

Note that this strategy allows us to use the cache and remember previously found paths. In the rare event that a trajectory is found, which is not represented in the current cache, we resort to DBF on the full time window, guaranteeing a valid cache for the current SSP iteration.

Summing up costs along paths including the dashed nodes in Fig. 4 may lead to negative entry costs and thus to shortest paths which overlap paths found in previous successive shortest path iterations yielding negative cycles in the residual graph. This indicates, that a previously deleted part in the graph of this trajectory should be changed to obtain the optimal solution. As we have collapsed the entry edges of the frame which has left the current time window, this subtrajectory is not part of the graph anymore, and an approximation must be made. We do so by prohibiting the new shortest path from containing the source or the sink node except for the first or the last node of the path. Note that this is a reasonable assumption (which is automatically fulfilled by the optimal BF/DBF/oDBF algorithms) as trajectories can only start or end at the source or sink nodes, respectively. For the memory-bounded case, our algorithm thus simply returns the next best shortest path which fulfills these requirements.
Algorithm 4: Data association by successive shortest path using mboDBF

Input: Detections for time step $t$, $X_t = \{x_i\}$,
Graph $G(f)$, Cache $C$

Output: Set of trajectory hypotheses $T = \{T_k\}$

1. foreach node $\in [t_{max} - \tau]$ do
   2. // remember predecessor
      3. $G(f) \leftarrow$ UpdateEntryEdge($G(f)$)
   4. $G(f) \leftarrow$ RemoveObservations($G(f), t_{max} - \tau$)
   5. // From here on, mboDBF and oDBF are identical.
   6. $G_r(f) \leftarrow G(f)$ // initialize $G_r(f)$ as a DAG
   7. $\pi_0 \leftarrow C(t - 1, k = 0)$
   8. $\gamma_0, \pi_0 \leftarrow$ DAG-SP($G_r(f)$)
   9. $q \leftarrow \emptyset$ // $q$ is maintained for every iteration $k$
   10. while $C(\gamma_k) < 0$ do
       11. $\delta_i \leftarrow$ FindMostRecentCache($\gamma_{k-1}, C$)
       12. $\gamma_{k+1} = \gamma_{k+1}$, $\delta_i \in \Delta$
       13. $\pi_k \leftarrow C(\delta_i, k)$ // predecessor map $\pi_k$ must be updated
       14. $\gamma_k \leftarrow C(\delta_i, k)$ // $\pi_k$ for $\gamma_{k-1} = \delta_i \gamma_{k-1}$ is invalid
       15. $q \leftarrow oDBFQueue(q, \pi_k, G_r(f))$
   16. return $T$

5. Experimental Evaluation

We evaluate our algorithms on the challenging KITTI [19] and PETS 2009 [15] tracking benchmarks. For object detection, we make use of the deformable part-based model (“DPM”) [17] in combination with the pre-trained object detection model provided with the KITTI benchmark. We convert the detector score for each bounding box $d_i$ into the unary cost $C_i$ using logistic regression $C_i = 1/(1 + e^{\beta d_i})$. To encode association costs, we use five different pairwise similarity features $s = \{s_i\}$: bounding box overlap, color histogram similarity, cross-correlation, bounding box size similarity, and location similarity which have been proposed in the literature [20][3][18]. Similar to detection, here, $s$ denotes the output of a logistic function which has been learned via logistic regression from training data and ranges $[0, 1]$. The detection/association cost for each edge $(u, v)$ is then defined as $C_{u,v} = ((1 - s) + o)^T w$, where $o$ denotes an offset and $w$ the scale. Note that the offset is required to allow for negative as well as positive costs. All parameters $\beta, o, w$ have been obtained using grid search on the respective training sets and kept fix to the following values during our experiments: $w = [5.0, 5.0, 10.0, 0.0, 0.0, 2.5, 1.0, 6.75, 5.75, 0.0]^T$, $o = [0.0, 0.0, -1.0, 0.0, -0.5, -0.5, -0.5, -0.5, -0.5, -0.25]^T$, $\beta = 15$. We refer the reader to the supplementary material for additional results and videos.

Comparison to State-of-the-art on KITTI: We first compare the proposed DBF and mboDBF algorithm against three baselines [18, 31, 3] as well as the pairwise optimal Hungarian method (“HM”) on the challenging KITTI dataset (see Fig. 7 for an illustration). The metrics we use are described in [27][8]. As shown in Table 1 our optimal DBF algorithm outperforms current state-of-the-art. In our experiments, we made use of a relatively low threshold $d_i = -0.3$ for the object detector to avoid early pruning and evaluate each method with respect to outlier rejection performance. Note that our method attains the best performance with respect to mostly tracked trajectories (“MT”) while only exhibiting a slightly higher false alarm rate (“FAR”) than the other methods. Also note the little loss in performance when running mboDBF for a window length of $\tau = 10$. Compared to the non-optimal DP solution, mboDBF achieves higher performance, especially in terms of identity switches and fragmentations.

Comparison to State-of-the-art on PETS2009: We additionally evaluate our method on the commonly used PETS2009 dataset for sequence “S2.L1”. We used the detections and ground truth provided by the authors of [29] and therefore only compare MOTA. For reference, we also state the other metrics. While our

|       | HM [3] | DP [31] | BS [3] | mboDBF | DBF |
|-------|--------|---------|--------|--------|-----|
| MOTA  | 0.90   | 0.89    | 0.89   | 0.82   | 0.75|
| MOTP  | 0.81   | 0.81    | 0.56   | 0.56   | 0.60|
| MODA  | 0.90   | 0.89    | 0.89   | 0.89   | 0.89|
| MODP  | 0.81   | 0.81    | 0.57   | 0.57   | 0.60|
| MT    | 0.89   | 0.58    |       |       |     |
| ML    | 0.0    | 0.0     |       |       |     |

Table 2. Comparison of our proposed method to three baselines on PETS2009 [15].
DBF implementation outperforms current state-of-the-art, our memory bounded version (mboDBF) performs on par with the baselines.

Comparing Min Cost Flow Solutions: In this section, we compare our globally optimal methods (DBF, oDBF) as well as our approximative mboDBF algorithm for window size $\tau = 10$ against Bellman-Ford as well as the non-optimal DP solution [13] in terms of relaxations and runtime. For Bellman-Ford, we use the textbook implementation described [14], and refer to it as “BF”. For a fair comparison, we implement all our solvers in Python using the same data structures. As for DP we only have access to a Matlab implementation, we only report results in terms of relaxations. Fig. 5 depicts relaxations and execution time as a function of the number of frames for all KITTI test sequences. Note that the worst case complexity of BF$n^2$ is never reached by the early terminating version (BF), which is about one order of magnitude faster. This is expected as the graph is not fully connected and forward pointing edges dominate the graph structure even for residual graphs. In turn, our globally optimal dynamic solvers (DBF, oDBF) outperform BF by more than one order of magnitude. Furthermore, we note that oDBF is half an order of magnitude faster than DBF as it only requires more computations when the shortest path is extended in the current time step. When comparing our optimal oDBF to the approximative DP solution of [31], we find that the proposed approach carries out slightly more relaxations while still being optimal. In contrast, our memory-bounded solver mboDBF outperforms DP by approximately one order of magnitude while at the same time being more accurate (see Table 1). Importantly, unlike any other solver, its complexity is constant and not affected by the size of the sequence. Additionally, it can be applied to unlimited data streams as its memory requirements are also constant. For very short sequences mboDBF results in a slight overhead due to the requirement of maintaining an up-to-date cache.

Sliding Window Size of mboDBF: Finally, we evaluate the tracking performance of mboDBF for different values of $\tau$ (c.f. Table 3 and Fig. 6) on all frames of the KITTI training dataset (the test set ground truth is not publicly available). The run time comparison shows that for a reasonable value of $\tau = 10$, our non-optimized Python implementation of mboDBF requires less than 10ms which is sufficient for many real-time online applications.

![Figure 5. Comparing Relaxations (top) and Run Time (bottom).]  
We compared all solvers using the KITTI training set. We use a window size of $\tau = 10$ for mboDBF and evaluate all methods on a fully connected network without pruning any edges.

| history length | 5    | 10   | 15   | 20   | 50   | 100  |
|----------------|------|------|------|------|------|------|
| MOTA           | 0.50 | 0.51 | 0.48 | 0.49 | 0.51 | 0.52 |
| MOTP           | 0.78 | 0.78 | 0.78 | 0.78 | 0.78 | 0.78 |
| FAR            | 0.18 | 0.19 | 0.19 | 0.19 | 0.20 | 0.19 |
| MT             | 0.15 | 0.18 | 0.17 | 0.18 | 0.19 | 0.18 |
| ML             | 0.30 | 0.31 | 0.34 | 0.33 | 0.30 | 0.26 |
| Id-Switches    | 56   | 55   | 46   | 47   | 52   | 54   |
| Fragmentations | 328  | 333  | 303  | 321  | 351  | 371  |

Table 3. mboDBF tracking performance on the KITTI training set for different values of the sliding window $\tau$.

6. Conclusions

In this paper we have presented a computational efficient dynamic approach to solve data association problems which are formulated as a min-cost flow network. We showed that our algorithms significantly reduce the number of relaxations required to solve the problem and proposed a memory-bounded approximation which handles the online scenarios that couldn’t be dealt with before while still reaching state of the art performance. In future work we plan to extend our approach to explicitly tackle occlusions, incorporate additional high-level features, e.g., map information, and consider the association problem for all relevant object classes as a joint optimization problem.
Figure 6. **Comparing Relaxations (left) and Run Time (right).** We evaluated the complexity of mboDBF for different values of its history length $\tau$ on the whole KITTI training set.

Figure 7. **Example Results on the KITTI Benchmark.** We show images 508, 514, 566, 601, 609 and 635 of sequence 20 from the training set using our DBF algorithm.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.

[2] A. Andriyenko and K. Schindler. Multi-target tracking by continuous energy minimization. In *CVPR*, 2011.

[3] A. Andriyenko, K. Schindler, and S. Roth. Discrete-continuous optimization for multi-target tracking. In *CVPR*, 2012.

[4] C. Arora and A. Globerson. Higher order matching for consistent multiple target tracking. In *ICCV*, 2013.

[5] B. Benfold and I. Reid. Stable multi-target tracking in real-time surveillance video. In *CVPR*, 2011.

[6] J. Berclaz, F. Fleuret, and P. Fua. Multiple object tracking using flow linear programming. In *Proceedings of the 12th IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (Winter-PETS)*, pages 1–8, 2009.

[7] J. Berclaz, F. Fleuret, E. Turetken, and P. Fua. Multiple object tracking using k-shortest paths optimization. *PAMI*, 2011.

[8] K. Bernardin and R. Stiefelhagen. Evaluating multiple object tracking performance: The clear mot metrics. *IJIVP*, 1:1–10, 2008.

[9] E. Brau, J. Guan, K. Simek, L. D. Pero, C. R. Dawson, and K. Barnard. Bayesian 3d tracking from monocular video. In *ICCV*, 2013.

[10] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. J. V. Gool. Online multiperson tracking-by-detection from a single, uncalibrated camera. *PAMI*, 33(9):1820–1833, 2011.

[11] A. Butt and R. Collins. Multi-target tracking by lagrangian relaxation to min-cost network flow. In *CVPR*, 2013.

[12] W. Choi, C. Pantofaru, and S. Savarese. A general framework for tracking multiple people from a moving camera. *PAMI*, 2013.

[13] W. Choi and S. Savarese. A unified framework for multi-target tracking and collective activity recognition. In *ECCV*, 2012.

[14] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms.* McGraw-Hill Higher Education, 2001.

[15] A. Ellis and J. M. Ferryman. Pets2010 and pets2009 evaluation of results using individual ground truthed single views. In *AVSS*, pages 135–142. IEEE Computer Society, 2010.

[16] A. Ess, B. Leibe, K. Schindler, and L. V. Gool. Robust multi-person tracking from a mobile platform. *PAMI*, 31:1831–1846, 2009.

[17] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *PAMI*, 32:1627–1645, 2010.

[18] A. Geiger, M. Lauer, C. Wojek, C. Stiller, and R. Urtasun. 3d traffic scene understanding from movable platforms. *PAMI*, 2014.

[19] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *CVPR*, 2012.

[20] A. V. Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *J. Algorithms*, 32:1627–1645, 2010.

[21] B. S. Horesh, J. Berclaz, F. Fleuret, and P. Fua. Multi-Commodity Network Flow for Tracking Multiple People. *PAMI*, 2013.

[22] H. Idrees, I. Saleemi, and M. Shah. Statistical inference of motion in the invisible. In *ECCV*, 2012.

[23] J. F. P. Kooij, G. Englebienne, and D. M. Gavrila. A non-parametric hierarchical model to discover behavior dynamics from tracks. In *ECCV*, 2012.

[24] L. Leal-Taixé, G. Pons-Moll, and B. Rosenhahn. Everybody needs somebody: modeling social and group-
ing behavior on a linear programming multiple people tracker. In *ICCV Workshops*, 2011.

[25] L. Leal-Taixé, G. Pons-Moll, and B. Rosenhahn. Branch-and-price global optimization for multi-view multi-target tracking. In *CVPR*, 2012.

[26] B. Leibe, N. Cornelis, K. Cornelis, and L. Van Gool. Dynamic 3d scene analysis from a moving vehicle. In *CVPR*, 2007.

[27] Y. Li, C. Huang, and R. Nevatia. Learning to associate: HybridBoosted multi-target tracker for crowded scene. In *CVPR*, 2009.

[28] J. Liu, P. Carr, R. T. Collins, and Y. Liu. Tracking sports players with context-conditioned motion models. In *CVPR*, 2013.

[29] A. Milan, K. Schindler, and S. Roth. Detection- and trajectory-level exclusion in multiple object tracking. In *CVPR*, 2013.

[30] D. Mitzel and B. Leibe. Taking mobile multi-object tracking to the next level: People, unknown objects, and carried items. In *ECCV*, 2012.

[31] H. Pirsiavash, D. Ramanan, and C. C. Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *CVPR*, 2011.

[32] A. V. Segal and I. Reid. Latent data association: Bayesian model selection for multi-target tracking. In *ICCV*, 2013.

[33] B. Song, T.-Y. Jeng, E. Staudt, and A. K. Roy-Chowdhury. A stochastic graph evolution framework for robust multi-target tracking. In *ECCV*, 2010.

[34] J. Xing, H. Ai, and S. Lao. Multi-object tracking through occlusions by local tracklets filtering and global tracklets association with detection responses. In *CVPR*, 2009.

[35] F. Xiong, O. I. Camps, and M. Sznaier. Dynamic context for tracking behind occlusions. In *ECCV*, 2012.

[36] B. Yang, C. Huang, and R. Nevatia. Learning affinities and dependencies for multi-target tracking using a crf model. In *CVPR*, 2011.

[37] B. Yang and R. Nevatia. An online learned crf model for multi-target tracking. In *CVPR*, 2012.

[38] B. Yang and R. Nevatia. Online learned discriminative part-based appearance models for multi-human tracking. In *ECCV*, 2012.

[39] S.-I. Yu, Y. Yang, and A. Hauptmann. Harry potter’s marauder’s map: Localizing and tracking multiple persons-of-interest by nonnegative discretization. In *CVPR*, 2013.

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

[41] L. Zhang and L. van der Maaten. Structure preserving object tracking. In *CVPR*, 2013.