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, resulting in a computationally and memory-bounded solution. First, we introduce a dynamic version of the successive shortest-path algorithm which solves the data association problem optimally while reusing computation, resulting in faster inference than standard solvers. Second, we address the optimal solution to the data association problem when dealing with an incoming stream of data (i.e., online setting). Finally, we present our main contribution which is an approximate online solution with bounded memory and computation which is capable of handling videos of arbitrary 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

In recent years the performance of object detectors has improved substantially, reaching levels which nowadays make them applicable to real-world problems. A notable example is face recognition where algorithms that employ deep learning features have attained human performance [39]. Many aspects have been key for the success of object recognition. The creation of benchmarks such as PASCAL [18] and ImageNet [36] made experimentation more rigorous and stimulated the development of key algorithms such as the deformable part-based model (DPM) [19] and more recently deep learning approaches based on convolutional nets such as [26] and R-CNN [23].

With the success of object detection came also the success of *tracking-by-detection* approaches [2, 10, 11, 32, 35, 43, 46] which track an unknown number of objects over time. The idea is to first detect objects 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. The simplest approach is to independently establish associations between all pairs of consecutive frames, a problem which can be solved optimally in polynomial time using the Hungarian method. However, in challenging situations ambiguities are hard to resolve using local information alone and early errors cannot be corrected at later frames leading to globally inconsistent tracks. More recently, sophisticated discrete-continuous optimization schemes have been proposed [2], 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 problems are highly non-convex, hard to optimize and provide no guarantees of optimality.

In their seminal work, Zhang *et al.* [46] showed how multi-frame data association can be cast as a network flow problem. The optimal solution can be 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 for all consecutive frames. Recent extensions [3, 6, 11, 35] demonstrate speedups by using more elaborate graph solvers, and show how higher-order motion models can be incorporated into the formulation at the cost of giving up global optimality.

In this paper we are interested in making min-cost flow solutions applicable to real-world scenarios such as autonomous driving. Towards this goal two main problems need to be addressed: (i) current approaches assume a batch setting, where all detections must be available a priori, and (ii) their memory and computation requirements grow unbounded with the size of the input video. As a consequence existing min-cost flow techniques cannot be employed in robotics applications where tracking algorithms are required to run non-stop in an online setting while competing with other processes for a limited computation and memory budget. In this paper we develop practical tracking by detection solutions that

- perform *computations only when necessary*,
- handle an *online* stream of data, and
- are *bounded in memory and computation*.

In particular, we show that the first two properties can be achieved while maintaining optimality, and an approximate solution is possible for the latter which still performs very well in practice. We demonstrate the effectiveness of the proposed solvers on the KITTI [22] and PETS 2009 [20] tracking benchmarks. As evidenced by our experiments, our approach is significantly more efficient than existing min-cost flow algorithms. Furthermore, a near-optimal solution is retrieved when bounding the memory and computation to as little as 20kB and 10ms. We make our code, dataset and supplementary material available on our project website[1].

2. Related Work

Multi-target tracking approaches can be divided into two main categories: filtering-based approaches and batch methods. *Filtering-based methods* [10, 17, 33, 34] are based on the Markov assumption, *i.e.*, the current state depends only on the previous states. 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 social group behavior in the case of pedestrian tracking [25, 27, 37]. Discrete-continuous optimization techniques have been proposed [2, 32], where discrete (data association) and continuous (trajectory fitting) optimization problems are solved in an alternating way until converging to a hopefully better local minimum. In [9, 13, 14, 44], 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 [28, 29, 43, 45]. In [5, 6], 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 [4, 12, 16, 30] by using context from outside the object region or by building strong statistical motion models.

[1]http://www.cs.toronto.edu/~boundTracking

While most of the aforementioned formulations resort to approximate optimization without optimality guarantees, Zhang *et al.* [46] showed how data association with pairwise energies can be formulated as a network flow problem such that standard graph solvers can be leveraged to retrieve the global optimum. Their formulation solves for the globally optimal trajectories including their number, and hence implicitly solves the model selection problem. To reduce the computational complexity of min-cost flow algorithms, [6, 35] proposed to use the successive shortest-path (SSP) algorithm as solver. Further speed-ups have been achieved in [35] using a greedy dynamic programming (DP) approximation. The min-cost flow idea has been extended in [3, 11] to include higher-order terms at the price of loosing optimality. More recently, Wang *et al.* [41] proposed to jointly model the appearance of interacting objects by integrating linear flow constraints into the framework.

While all of the existing min-cost flow formulations assume a batch setting, in this paper we propose a computationally and memory-bounded version of this algorithm, which is able to process video sequences frame-by-frame while reusing computation via efficient caching strategies. Furthermore, our scheduling strategy performs computations only when neccessary which also speeds up traditional batch solvers.

3. Review on Optimal Tracking-by-Detection

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 these detections. In this section, we briefly review the necessary preliminaries to our contributions in Section 4. In particular, we review how to formulate multi-target tracking as a min-cost flow problem and how to solve it using the successive shortest path algorithm.

3.1. Tracking as Min-Cost Flow

Following the tracking-by-detection paradigm, we assume that a set of detections $\mathcal{X} = \{\mathbf{x}_i\}$ is available as input. Let $\mathbf{x}_i = (x_i, t_i, s_i, a_i, d_i)$ 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 a sequence of observations $T_k = (o_1, o_2, \dots, o_{l_k})$ with $o \in \{1, \dots, |\mathcal{X}|\}$ the detection index of temporally adjacent detections $t_{o_{i+1}} = t_{o_i} + 1$.

The full association hypothesis is then given by a set of trajectories $\mathcal{T} = \{T_k\}$, and the data association problem can be formulated as a Markov random field (MRF). More specifically, we aim at maximizing the posterior probability of trajectories:

$$p(\mathcal{T}|\mathcal{X}) = p(\mathcal{T}) \prod_i p(\mathbf{x}_i|\mathcal{T}) \quad (1)$$

The observation model is given by

$$p(\mathbf{x}_i|\mathcal{T}) = \begin{cases} P_i & \text{if } \exists T_k \in \mathcal{T} \wedge i \in T_k \\ 1 - P_i & \text{otherwise} \end{cases} \quad (2)$$

where P_i denotes the probability of $\mathbf{x}_i$ being a true detection. The prior over trajectories decomposes into a product of unary and pairwise factors

$$p(\mathcal{T}) \propto \prod_{T \in \mathcal{T}} \Psi(T) \prod_{T,T' \in \mathcal{T}} [T \cap T' = \emptyset] \quad (3)$$

where the pairwise term ensures that trajectories are disjoint. The unary factors are given by

$$\Psi(T) = \Psi_{en}(\mathbf{x}_{o_1}) \Psi_{ex}(\mathbf{x}_{o_l}) \prod_{i=1}^{l-1} \Psi_{li}(\mathbf{x}_{o_i}, \mathbf{x}_{o_{i+1}}) \quad (4)$$

where $\Psi_{en}(\mathbf{x}_{o_1})$, $\Psi_{ex}(\mathbf{x}_{o_l})$ and $\Psi_{li}(\mathbf{x}_{o_i}, \mathbf{x}_{o_{i+1}})$ model the likelihood of entering a trajectory, exiting a trajectory and linking temporally adjacent detections within a trajectory.

Taking the negative logarithm of (Eq. 1), the maximization can be transformed into an equivalent minimization problem over flow variables [46] as follows

$$\begin{aligned} \mathbf{f}^* = \operatorname*{argmin}_{\mathbf{f}} & \sum_i C_i^{en} f_i^{en} + \sum_i C_i^{ex} f_i^{ex} \\ & + \sum_{i,j} C_{i,j}^{li} f_{i,j}^{li} + \sum_i C_i^{det} f_i^{det} \\ s.t.\ \ f_i^{en} + & \sum_j f_{j,i}^{li} = f_i^{det} = f_i^{ex} + \sum_j f_{i,j}^{li} \quad \forall i \end{aligned} \quad (5)$$

where $C_i^{en} = -\log \Psi_{en}(\mathbf{x}_i)$ is the cost of creating a new trajectory at $\mathbf{x}_i$ and $C_i^{ex} = -\log \Psi_{ex}(\mathbf{x}_i)$ is the cost of exiting a trajectory at $\mathbf{x}_i$. The cost of linking two consecutive detections $\mathbf{x}_i$ and $\mathbf{x}_j$ is denoted $C_{i,j}^{li} = -\log \Psi_{li}(\mathbf{x}_i, \mathbf{x}_j)$ and C_i^{det} encodes the cost of $\mathbf{x}_i$ being a true detection or a false positive (data term). Furthermore, the binary flow variables f_i^{en} or f_i^{ex} take value 1 if the solution contains a trajectory such that $\mathbf{x}_i$ is the first frame or last frame, respectively. $f_i^{det} = 1$ encodes the fact that $\mathbf{x}_i$ is part of a trajectory and $f_{i,j}^{li} = 1$ if a tracklet exists which contains both detections $\mathbf{x}_i$ and $\mathbf{x}_j$ in two consecutive frames.

In their seminal work, Zhang and Nevatia [46] showed how to map the problem in Eq. 5 into a min-cost flow network problem. Fig. 1 illustrates one such network graph, 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 $c(u_i, v_i) = C_i^{det}$ and flow $f(u_i, v_i) = f_i^{det}$. For entering, edges are introduced between the source s and the first node of each detection with cost $c(s, u_i) = C_i^{en}$ and flow f_i^{en}. For exiting, edges are introduced between the last node of each detection and the sink with cost $c(v_i, t) = C_i^{ex}$ and flow f_i^{ex}. Finally, edges between consecutive detections (v_i, u_j) encode pairwise association scores with cost $c(v_i, u_j) = C_{i,j}^{li}$ and flow $f_{i,j}^{li}$. While we assume that only detections in consecutive frames can be linked, this can be easily generalized by allowing transitions between detections in non-consecutive frames resulting in additional edges.

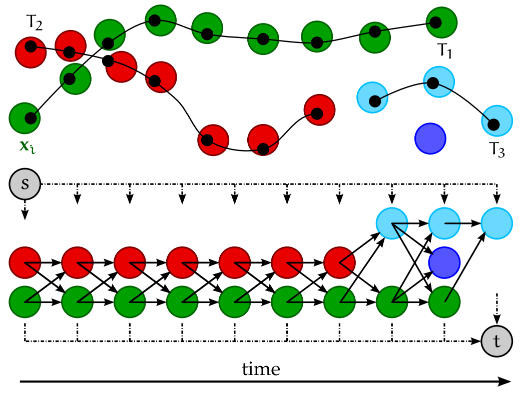


Figure 1: The original problem (top) is mapped into a min-cost flow network (bottom). Ground truth trajectories are shown in black. Colored nodes encode detections and correspond to two nodes in the min-cost flow network G. For clarity of illustration, edges from the source and to the sink have been ommitted.

To find the optimal solution, Zhang *et al.* [46] start with flow zero, and augment the flow one unit at a time, applying the push relabeling algorithm [24] 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 $\mathcal{O}((mn^2 \log^2(n)))$, with n the number of nodes and m the number of edges.

3.2. Successive Shortest-Path (SSP)

Recently, [6, 35] proposed to replace the costly push relabeling algorithm by the successive shortest-path (SSP) algorithm [1, p. 104]. This reduces the computational complexity to $\mathcal{O}(K(n\log(n) + m))$ where n is the number of nodes, m is the number of edges and K denotes the number of trajectories which is upper bounded by the number of nodes n. This section gives a brief introduction to the SSP algorithm as it forms the foundation for our contributions. We refer the reader to the supplementary material for technical details in greater depth and an illustrative example.

The SSP algorithm works as follows: it first computes the shortest path between source and target (*i.e.*, path with the lowest negative cost). It then iterates between reversing the edges of the previously found shortest path to form the residual graph G_r, and computing the shortest path in this new residual graph. This process is iterated until no trajectory with negative cost can be found. Finally, trajectories are extracted by backtracking connected, inverted edges starting at the target node.

In the first iteration the shortest path from the source to the target is efficiently retrieved using dynamic programming [15, p. 655] as the input graph is directed and acyclic. In the following iterations, Dijkstra's algorithm can be leveraged after converting the graph such that all costs are positive. This conversion is achieved by simply replacing the current cost $C_{u,v}$ between nodes (u, v) linked by a directed edge, with $C'_{u,v} = C_{u,v} + d(u) - d(v)$, where $d(u)$ and $d(v)$ are the distance on the shortest path from the source to nodes u and v (*c.f.* supplementary material). Importantly, for u and v on the shortest path we have $C'_{u,v} = 0$ after conversion. Note that for graphs with positive costs and unit flow capacity, the SSP algorithm is similar to the k-shortest paths ("KSP") algorithm [38].Furthermore, one could use other algorithms such as Bellman-Ford to replace Dijkstra and handle negative costs directly. However, this would result in worst complexity, *i.e.*, $\mathcal{O}(K(n^2))$.

4. Dynamic Online and Bounded Tracking

In this section we present new algorithms that have the necessary properties for tracking-by-detection to be applicable to real-world scenarios: dynamic computation, handling online data, and bounded memory and computation. We start by proposing a distributed optimal SSP algorithm which leverages a dynamic priority cue, saving computation when compared to the original SSP algorithm. We then extend this algorithm to handle an online data stream. Finally, we proposed our main contribution, a memory and computation bounded online SSP algorithm.

4.1. Optimal Dynamic Min-Cost Flow (dSSP)

In this section we propose a novel *dynamic* algorithm which performs computations only when necessary. Dijkstra's algorithm 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. Nodes are held in a priority queue, guaranteeing that the most promising node is relaxed in each iteration [2]. This priority queue is initialized to hold only the source node, and upon convergence all nodes have been visited.

The key intuition behind our dynamic computation is that calling Dijkstra at each iteration of the SSP algorithm is suboptimal in terms of runtime as from one iteration to the next only a small part of the graph has changed (*i.e.*, only the reversed edges are different). Inspired by the distributed Bellman-Ford algorithm [8, 40], we propose to reuse computation and only update the predecessors when needed. This can be easily implemented within Dijkstra by using a *dynamic priority queue*, which contains only nodes that have changed. Initially, this queue comprises the changes introduced by reducing edges after finding the previous shortest path in order to handle positive costs as in KSP. The queue is then dynamically updated depending on which successors require changes. We refer the reader to Algo. 1 for a summary of our dynamic algorithm.

We illustrate our dynamic algorithm within the computation of a single iteration of SSP in Fig. 2 for an example containing four frames with three detections each. Given the shortest path found in the previous iteration, we revert its edges to form the residual graph in Fig. 2(a). Note that the corresponding predecessor maps 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. 2(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 to their successors when taken from the priority queue. All successors which receive an update, *i.e.*, for which the predecessor has changed, are themselves added to the priority queue (indicated by the red edges in Fig. 2(b)). Note that those nodes are not necessarily part of the previous shortest path but can be retrieved efficiently as the successors of the nodes in the current queue. This way, our algorithm recursively updates all required nodes and ensures global optimality. The algorithm (one iteration of SSP) terminates when the queue is empty as all shortest paths have been updated. Note that this dynamic computation is employed at each iteration of SSP, and SSP terminates when no new shortest path with negative cost can be found (Fig. 2(c)). The final trajectories are extracted by collecting all backward pointing edges of G_r, starting at the target node (Fig. 2(d)). In contrast to Dijkstra's original algorithm, our dynamic broadcasting scheme relaxes only parts of the graph for typical tracking networks as illustrated in Fig. 2(e) and demonstrated later in our experimental evaluation.

4.2. Optimal Online Solution (odSSP)

Next, we extend the dynamic min-cost flow formulation introduced in the previous section to the online setting. Our intuition is that every time a new observation arrives (*i.e.*, set of detections at time t), we would like to reuse computation from the shortest paths and trajectories computed in the previous time step which involve all detections up to time $t - 1$. This is possible as in the online setting, the new

[2]This is guaranteed as all costs are positive.

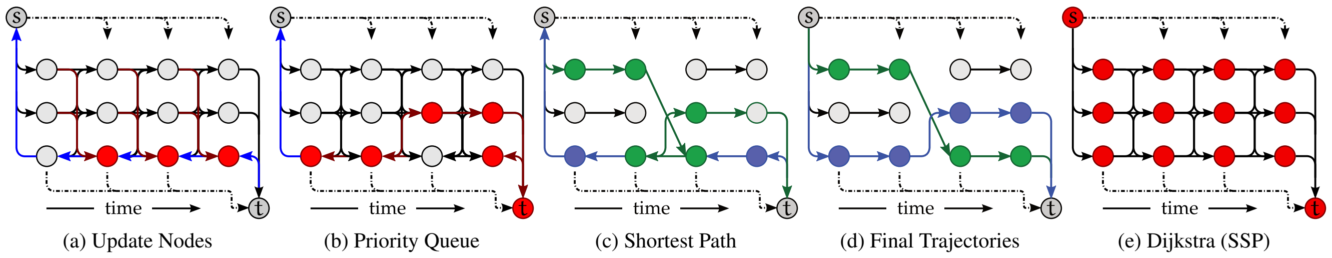


Figure 2: **Dynamic Message Broadcasting (dSSP):** For a given residual graph (shortest path in blue), (a) nodes with invalid predecessors which require an update (red) are detected and queued. (b) The queue is processed by successively taking nodes from the queue and relaxing their outgoing edges. Successors with updated predecessors (red) are added to the queue. Exploiting Dijkstra's algorithm, the priority queue is processed until no nodes are left. The algorithm terminates with the solution for the current residual graph (c)+(d). Trajectories are encoded by backward pointing edges (green and blue). While our dynamic broadcasting scheme exploits the intrinsic properties of the tracking problem and updates only deprecated predecessor maps (indicated by the red nodes in (a) and (b)), Dijkstra's original algorithm requires the full predecessor map and broadcasts messages to all nodes (indicated by the red nodes in (e)).

network contains the previous network and some additional edges/nodes. It is important to note that a naïve extension of trajectories would violate optimality if the new evidence (*i.e.*, detections at time t) changes the optimal trajectories for detections in previous time steps $(1, \cdots, t-1)$.

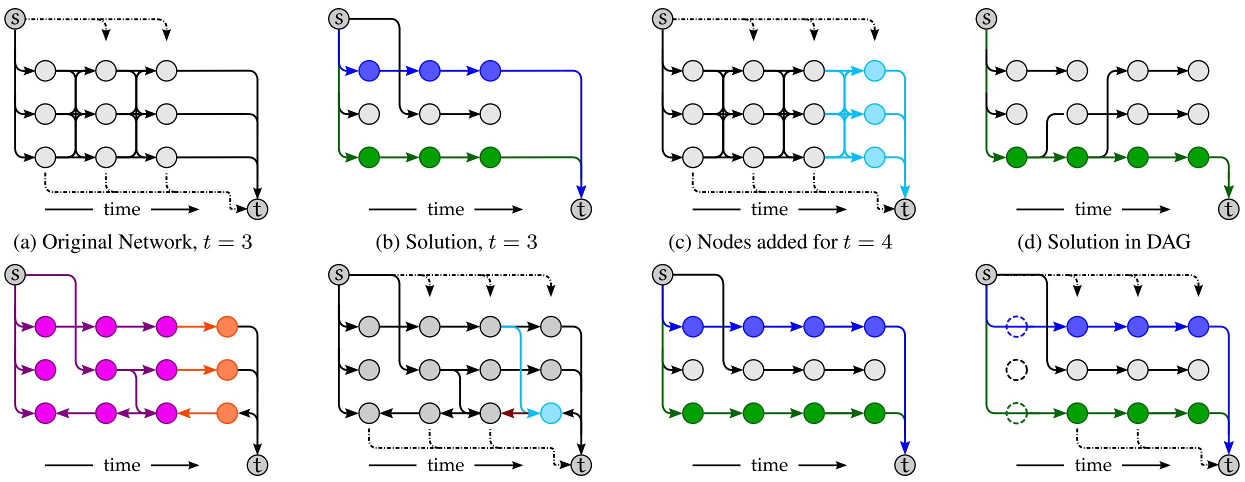
We illustrate our dynamic online algorithm with an example in Fig. 3. Consider the network specified in Fig. 3(a) where the most recent time step is $t = 3$ and the optimal set of trajectories and predecessor maps have been computed. Fig. 3(b) illustrates the shortest paths as well as the two optimal trajectories (in blue and green respectively) computed at time $t = 3$. As a new frame arrives, new nodes (depicted in cyan in Fig. 3(c) are added, extending the graph to the next time step. The first trajectory can be computed by applying dynamic programing only to the edges involving the new nodes, as in a DAG the predecessors do not change from previous time steps. For successive trajectories (next iterations of the SSP algorithm), the predecessor maps from the previous time steps can be reused only if the trajectories under consideration are currently in the same order as when applying SSP to the previous time frame $t-1$ (*i.e.*, before we received the new observation). This will not be the case when we have competing trajectories with similar costs, as given the new evidence (new frame) the ordering of these trajectories is likely to change. To handle this case, we utilize a caching strategy which keeps all predecessor maps for a cache of $|\mathcal{C}|$ frames in memory. Similar to the offline strategy, the online version implements a dynamic priority queue. Initially, this queue comprises the changes introduced by the added nodes and edges for the most recent time step. The queue is then dynamically updated depending on which successors require changes.

Coming back to our example, the optimal solution for the first iteration of SSP is found in Fig. 3(d). The predecessor maps for the new nodes (orange nodes in Fig. 3(e)) have to be re-estimated only if they were part of the shortest path. Towards this goal, we employ our dynamic broadcasting strategy where only the node in cyan in Fig. 3(f) is put into the initial priority queue. As in this example the optimal trajectories are consistent with the ones from the previous time instant (see Fig. 3(b) and Fig. 3(g)), the optimal trajectories can be computed very efficiently, resulting in massive computational gains.

4.3. Memory-Bounded Solution (mbodSSP)

While the odSSP algorithm handles the online setting, it does not scale to very large problems since messages might broadcast back to very early frames in order to guarantee optimality. Furthermore, memory requirements still grow unbounded with the length of the sequence. Thus, these algorithms are not applicable to autonomous driving or surveillance scenarios.

In this section we propose a memory and computationally bounded approximation which we call "mbodSSP". The key intuition is that given some computation and memory budget, we can safely neglect most of the past, and only retain the interesting information, *i.e.*, the trajectories that were optimal. We refer the reader to Algo. 2 for a depiction of the algorithm, and focus on explaining it through an example. Consider the case where we assume a budget of $\tau = 4$ frames, and the solution from the previous time step is given in Fig. 3(g). Before adding new nodes to the graph for the next time step $t+1$, we remove the oldest time step $t-\tau$ from the graph as illustrated in Fig. 3(h) to maintain the memory/computation constraints. Simply deleting edges from the graph is suboptimal as it completely discards computations from previous time steps (before the new observation arrives). In order to "remember" known paths, for each trajectory we sum the cost of the predecessor node at time $t-\tau$ to the corresponding successor at time $t-\tau+1$.



(a) Original Network, $t = 3$ (b) Solution, $t = 3$ (c) Nodes added for $t = 4$ (d) Solution in DAG

(e) Merge Solution+Cache (f) Broadcast Changes (g) Decoded Trajectories (h) Clip Graph (mbodSSP)

Figure 3: **Online (a)-(g) and Memory Bounded (h) Algorithm:** Assume that for the original network at $t = 3$ (a), the solution (b) is already known (shortest paths in blue and green). A new set of detections (cyan 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 maps from the previous (magenta nodes) and current time step (orange nodes) are merged (e). The queue initially contains nodes with outdated predecessors from the last shortest path (cyan). The priority queue is processed until convergence (Fig. 2) and the predecessors are updated (f). In this example, the algorithm terminates after one iteration and the optimal solution is found (g). For the mbodSSP algorithm, paths which pass through nodes outside the history window are merged with the entry costs of the next time step (h).

This strategy allows us to use the cache and remember previously found paths. In the rare event that a trajectory which is not represented in the current cache is found, we resort to odSSP on window $[t - \tau + 1, t + 1]$, guaranteeing a valid cache for the current SSP iteration. The cached predecessor maps are clipped using the same strategy. While optimality is violated as no changes can be applied to paths beyond $t - \tau + 1$, our experiments demonstrate that for many cases of practical interest small windows are sufficient to obtain near-optimal solutions. In particular, our memory bounded approximation is able to maintain track ids over periods much longer than the window itself. In contrast, when splitting the sequence into batches and applying the min-cost flow network algorithm separately to each batch, ad-hoc heuristics are required to resolve this problem.

5. Experimental Evaluation

We evaluate our algorithms on the challenging KITTI [22] and PETS 2009 [20] tracking benchmarks. For PETS we use the object detections provided by Andriyenko *et al.* [2]. For KITTI we compare the DPM reference detections [19] provided on the KITTI website[3] with the recently proposed Regionlet detections [42] provided to us by the authors.

We convert the detection score of each bounding box d_i into a unary cost value C_i using logistic regression $C_i = 1/(1 + \mathrm{e}^{\beta d_i})$ on the training set. To encode association costs, we use six different pairwise similarity features $\mathbf{s} = \{s_l\}$: bounding box overlap, orientation similarity, color histogram similarity, cross-correlation, location similarity, and optical flow overlap. Similar to the detection scores we pass the association features through a logistic function using logistic regression yielding $\mathbf{s} \in [0, 1]^6$. The detection/association cost for each edge (u, v) is then defined as $C_{u,v} = ((\mathbf{1} - \mathbf{s}) + \mathbf{o})^\top \mathbf{w}$, where $\mathbf{o}$ denotes an offset and $\mathbf{w}$ the scale. Note that the offset is required to allow for negative as well as positive costs. All parameters $(\mathbf{o}, \mathbf{w})$ have been obtained using block coordinate descent on the respective training sets and kept fix during all our experiments. We refer the reader to the supplementary material for further details on the parameter setting, additional results, and videos.

Comparison to State-of-the-art on KITTI: We first compare the proposed dSSP and mbodSSP algorithms against four state-of-the-art baselines [2, 21, 31, 35] as well as the pairwise optimal Hungarian method ("HM") on the challenging KITTI dataset using the DPM reference detections [19]. The metrics we use are described in [7, 29]. As shown in Table 1 (left part), the optimal algorithm ("SSP") outperforms all other methods. Note that all discussed opti-

[3]http://www.cvlibs.net/datasets/kitti/eval_tracking.php

```
Algorithm 1: dSSP
   Input: Set of Detections $\mathcal{X} = \{\mathbf{x_i}\}$
   Output: Set of trajectory hypotheses $\mathcal{T} = \{T_k\}$
 1 $G(V,E,C,f) \leftarrow$ ConstructGraph($\mathcal{X}, s, t$)
 2 $f(G) \leftarrow 0$                                  // initialize flow to 0
 3 $\gamma_0, \pi_0 \leftarrow$ DAG-SP($G(f)$)          // shortest path in DAG
 4 $G_r^{(0)}(f) \leftarrow$ ConvertEdgeCosts($G(f), \gamma_0, \pi_0$) $G_r^{(1)}(f) \leftarrow$
   ComputeResidualGraph($G_r^{(0)}(f), \pi^{(0)}$)
 5 $q \leftarrow \emptyset$                    // $q$ is maintained for every iteration $k$
 6 while 1 do                                  // find shortest paths for $k \geq 1$
 7     $k \leftarrow k+1$
 8     $\pi^{(k)} \leftarrow \pi^{(k-1)}$
 9     $q \leftarrow \gamma^{(k-1)}$
       // process queue in time direction
10     foreach node $u \in q$ do
           // check predecessor from past
11         $\pi^{(k)} \leftarrow$ Update($\pi^{(k)}$, $u$)
12         if updated then
13             $q \leftarrow$ AddSuccessors($q$, $u$, $G_r^{(k)}(f)$)
       // process queue
14     while $\neg$ empty($q$) do
15         $u \leftarrow$ NodeWithMinDistance($q$)      // pop node
16         $q \leftarrow q \setminus u$
17         foreach node $v \in$ Successors($G_r^{(k)}(f)$,$u$) do
               // Check predecessor of $v$
18             $\pi^{(k)}(v) \leftarrow$ Relax($u$,$v$,$c$)
19             if $d(v) > d(u) + c(u,v)$ then
20                 $d(v) \leftarrow d(u) + c(u,v)$
21                 $q \leftarrow$ AddNode($q$, $v$)
22     $G_r^{(k)}(f) \leftarrow$ ConvertEdgeCost($G_r^{(k)}(f), \pi^{(k)}$)
23     $G_r^{(k+1)}(f) \leftarrow$ ResidualGraph($G_r^{(k)}(f), \gamma^{(k)}$)
       // evaluate converted costs
24     if $\sum_{i=1}^{k} \mathrm{cost}(\gamma^{(i)}) > |\mathrm{cost}(\gamma^{(0)})|$ then
25         break
26 return $\mathcal{T}$
```

```
Algorithm 2: mbodSSP
   Input: Current Detections $\mathcal{X}_t = \{\mathbf{x}_t\}$,
          Graph $G(V,E,C,f,s,t)$, Cache $\mathcal{C}$
   Output: Set of trajectory hypotheses $\mathcal{T} = \{T_k\}$
 1 if memorybounded then
 2     foreach node $u \in [t-\tau]$ do
           //remember predecessor by updating $c_{en,i}$
 3         $G(f) \leftarrow$ UpdateEntryEdge($G(f), u$)
 4         $G(f) \leftarrow$ RemoveObservation($G(f), u$)
 5 $G(f) \leftarrow$ AddObservations($G(f), \mathcal{X}_t$)          //still a DAG
 6 $\pi^{(0)} \leftarrow \mathcal{C}^{(0)}(t-1)$
   // run DAG-SP for edges $(u,v) \in [t-1,t]$
 7 $\gamma^{(0)}, \pi^{(0)} \leftarrow$ DAG-SP($G_r(f), t-1$)
 8 $G_r^{(0)}(f) \leftarrow$ ConvertEdgeCosts($G(f), \gamma^{(0)}, \pi^{(0)}, t-1$)
 9 $G_r^{(1)}(f) \leftarrow$ ComputeResidualGraph($G_r^{(0)}(f), \pi^{(0)}$)
10 $q \leftarrow \emptyset,\ k \leftarrow 0$
11 while 1 do
12     $k \leftarrow k+1$
       // $\gamma_{t-\delta_i}^{(k-1)} = \gamma_t^{(k-1)}, \delta_i \in \{0,\ldots,|\mathcal{C}|\}$
13     $\delta_i \leftarrow$ MostRecentCache($\mathcal{C}, \gamma^{(l)}\ \forall\ l = 0,\ldots,k-1$)
14     $\pi^{(k)} \leftarrow \mathcal{C}(\delta_i, k)$           // update predecessor map $\pi_k$
15     $q \leftarrow \{\gamma_t^{(k-1)}, (u,v) \in [t-\delta_i, t]\}$
       // Algo. 1, line 10
16     $G_r^{(k+1)}(f), \gamma^{(k)} \leftarrow$ ProcessQueue($q$, $\pi^{(k)}$, $G_r^{(k)}(f)$)
17 return $\mathcal{T}$
```

	HM	[2]	[31]	[35]	[21]	mbodSSP	SSP	mbodSSP*	SSP*
MOTA	0.42	0.35	0.48	0.44	0.52	0.52	**0.54**	0.67	0.67
MOTP	0.78	0.75	0.77	0.78	0.78	0.78	0.78	0.79	0.79
F1	0.60	0.61	0.67	0.62	0.69	**0.70**	**0.71**	0.83	0.83
FAR	**0.048**	0.46	0.18	0.053	0.083	0.14	0.11	0.34	0.40
MT	0.077	0.11	0.14	0.11	0.14	**0.15**	**0.21**	0.34	0.41
ML	0.42	0.34	0.34	0.39	0.35	**0.30**	**0.27**	0.10	0.090
IDS	12	223	125	2738	33	**0**	**7**	117	194
Frag.	578	624	**401**	3241	540	708	717	894	977

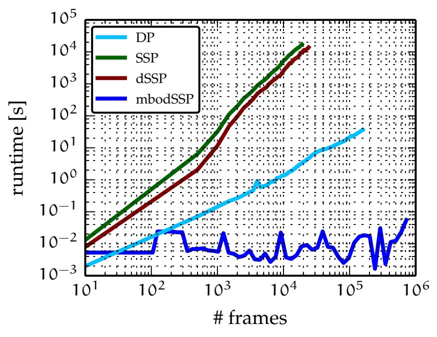
Table 1: Comparison of our proposed methods to four state of the art methods and a HM baseline implementation on KITTI-Car using the DPM reference detections and Regionlet detections (marked with a star).

	[2]	EKF [31]	[31]	mbodSSP	SSP
MOTA	**0.96**	0.68	0.91	0.89	0.91
MOTP	0.79	0.77	0.80	**0.87**	**0.87**
MT	**0.96**	0.39	0.91	0.89	0.89
ML	**0.0**	0.04	0.04	**0.0**	**0.0**
ID-switches	10	25	11	**7**	23
Fragmentations	8	30	**6**	100	100

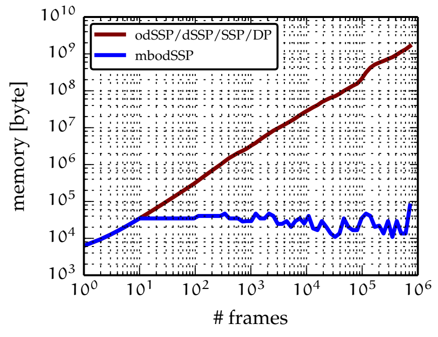
Table 2: Comparison of our proposed method to three baselines on PETS 2009 sequence "S2.L1".

mal batch solvers obtain identical tracking results, but with different run time thus we only state results for SSP. 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. The method of [2] struggled with the presence of outliers and $d_i = 0.0$ was used to obtain meaningful results. Also note the little loss in performance when running mbodSSP for a window length of $\tau = 10$ as a good trade-off between runtime and performance (Fig. 3). Compared to the non-optimal DP solution [35], mbodSSP achieves higher performance, especially in terms of identity switches and fragmentations. We provide additional qualitative results in the supplementary material.We also experiment with the detector of [42], which yields better results particularly for occluded objects. The increasing tracking performance indicates that with a good enough detector our approach could be used in practical applications.

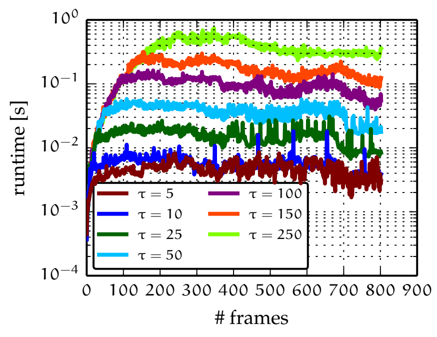
Comparison to State-of-the-art on PETS2009: We additionally evaluate our methods on the commonly used



(a) Run Time



(b) Memory Consumption



(c) Impact of History Length τ

Figure 4: **Run Time and Memory Comparison.** We compare computational performance of all solvers using one long sequence without ground truth annotations. This figure shows the mean runtime (a) and idealized memory consumption (b) for every solver. Additionally, we show the impact of different values for the history length τ (c).

τ	5	10	15	20	50	100
MOTA	0.50	**0.52**	0.48	0.49	0.51	**0.52**
MOTP	**0.78**	**0.78**	**0.78**	**0.78**	**0.78**	**0.78**
F1	0.69	0.70	0.68	0.69	0.70	**0.71**
FAR	0.15	**0.14**	**0.14**	**0.14**	**0.14**	**0.14**
MT	0.14	0.15	0.16	0.17	**0.18**	0.15
ML	0.30	0.30	0.34	0.33	0.30	**0.26**
IDS	2	**0**	**0**	**0**	4	5
Frag.	703	708	**690**	701	710	712

Table 3: Variation of the approximation parameter τ.

PETS2009 dataset for sequence “S2.L1”. We used the detections and ground truth provided by the authors of [2,31]. Both, the optimal algorithms and the memory-bounded approximation perform on par with current state-of-the-art. In particular performance for precision-based measures is notably good. Note that we used the same parameters as for the results presented on KITTI.

Comparing Min Cost Flow Solutions: We compare our globally optimal methods (dSSP, odSSP) as well as our approximate mbodSSP algorithm for window size $\tau = 10$ against a regular SSP implementation using Dijkstra’s algorithm (as described in [15]) as well as the non-optimal DP solution of [35] in terms of run time and memory consumption. For a fair comparison, we implement all our solvers in Python using the same data structures. Fig. 4 (a)-(b) depict execution time and memory consumption as a function of the number of frames for a very long sequence on KITTI. Note that our globally optimal dynamic solver (dSSP) outperforms the regular Dijkstra implementation (SSP) by a factor of 3 on average. To verify the correctness of our implementation against the optimal solution, we have used thousands of random graphs. Importantly, our experiments validate that the time complexity of mbodSSP is independent of the sequence length. Thus, it outperforms DP [35] in memory, computation as well as accuracy (see Table 1). A run time evaluation for KITTI including a comparison for the different training scenarios is discussed in greater depth in the supplementary material.

Sliding Window Size of mbodSSP: We evaluate the tracking performance of mbodSSP for different values of τ on KITTI. As shown in Fig. 4(c), for a value of $\tau = 10$, our non-optimized Python implementation of mbodDijkstra requires less than 10ms which is sufficient for many real-time online applications.

6. Conclusions

In this paper we have proposed solutions to make the use of min-cost flow tracking by detection possible in real world scenarios. Towards this goal we have designed algorithms that are dynamic, can handle an online data stream and are bounded in memory and computation. We have demonstrated the performance of our algorithms in challenging autonomous driving and surveillance scenarios. In future work we plan to extend our approach to handle long-term occlusions and to incorporate additional features, *e.g.*, map information.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice Hall, 1993. 3

[2] A. Andriyenko, K. Schindler, and S. Roth. Discrete-continuous optimization for multi-target tracking. In *CVPR*, 2012. 1, 2, 6, 7, 8

[3] C. Arora and A. Globerson. Higher order matching for consistent multiple target tracking. In *ICCV*, 2013. 1, 2

[4] 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

[5] H. Ben Shitrit, J. Berclaz, F. Fleuret, and P. Fua. Multi-Commodity Network Flow for Tracking Multiple People. *PAMI*, 36(8):1614–1627, 2013. 2

[6] J. Berclaz, F. Fleuret, E. Turetken, and P. Fua. Multiple object tracking using k-shortest paths optimization. *PAMI*, 33(9):1806–1819, 2011. 1, 2, 3

[7] K. Bernardin and R. Stiefelhagen. Evaluating multiple object tracking performance: The clear mot metrics. *JIVP*, 1:1–10, 2008. 6

[8] D. Bertsekas and R. Gallager. *Data networks*. Prentice-Hall, Inc., 1992. 4

[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. 2

[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. 1, 2

[11] A. Butt and R. Collins. Multi-target tracking by lagrangian relaxation to min-cost network flow. In *CVPR*, 2013. 1, 2

[12] X. Chen, Z. Qin, L. An, and B. Bhanu. An online learned elementary grouping model for multi-target tracking. In *CVPR*, 2014. 2

[13] 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

[14] R. T. Collins and P. Carr. Hybrid stochastic / deterministic optimization for tracking sports players and pedestrians. In *ECCV*, 2014. 2

[15] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001. 4, 8

[16] C. Dicle, O. I. Camps, and M. Sznaier. The way they move: Tracking multiple targets with similar appearance. In *ICCV*, December 2013. 2

[17] A. Ess, B. Leibe, K. Schindler, and L. V. Gool. Robust multiperson tracking from a mobile platform. *PAMI*, 31:1831–1846, 2009. 2

[18] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 88(2):303–338, 2010. 1

[19] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *PAMI*, 32:1627–1645, 2010. 1, 6

[20] J. Ferryman and A. Shahrokni. Pets2009: Dataset and challenge. In *PETS*, pages 1–6, 2009. 2, 6

[21] A. Geiger, M. Lauer, C. Wojek, C. Stiller, and R. Urtasun. 3D traffic scene understanding from movable platforms. *PAMI*, 36(5):1012–1025, 2014. 6, 7

[22] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *CVPR*, 2012. 2, 6

[23] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1

[24] A. V. Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *JA*, 22(1):1–29, 1997. 3

[25] J. F. P. Kooij, G. Englebienne, and D. M. Gavrila. A nonparametric hierarchical model to discover behavior dynamics from tracks. In *ECCV*, 2012. 2

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1

[27] L. Leal-Taixe, M. Fenzi, A. Kuznetsova, B. Rosenhahn, and S. Savarese. Learning an image-based motion context for multiple people tracking. In *CVPR*, 2014. 2

[28] L. Leal-Taixe, G. Pons-Moll, and B. Rosenhahn. Branch-and-price global optimization for multi-view multi-target tracking. In *CVPR*, 2012. 2

[29] Y. Li, C. Huang, and R. Nevatia. Learning to associate: HybridBoosted multi-target tracker for crowded scene. In *CVPR*, 2009. 2, 6

[30] J. Liu, P. Carr, R. T. Collins, and Y. Liu. Tracking sports players with context-conditioned motion models. In *CVPR*, 2013. 2

[31] A. Milan, S. Roth, and K. Schindler. Continuous energy minimization for multitarget tracking. *PAMI*, 36(1):58–72, 2014. 6, 7, 8

[32] A. Milan, K. Schindler, and S. Roth. Detection- and trajectory-level exclusion in multiple object tracking. In *CVPR*, 2013. 1, 2

[33] D. Mitzel and B. Leibe. Taking mobile multi-object tracking to the next level: People, unknown objects, and carried items. In *ECCV*, 2012. 2

[34] H. Nam, S. Hong, and B. Han. Online graph-based tracking. In *ECCV*, 2014. 2

[35] H. Pirsiavash, D. Ramanan, and C. C. Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *CVPR*, 2011. 1, 2, 3, 6, 7, 8

[36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *arXiv.org*, 1409.0575, 2014. 1

[37] X. Shi, H. Ling, W. Hu, C. Yuan, and J. Xing. Multi-target tracking with motion context in tensor power iteration. In *CVPR*, 2014. 2

[38] J. W. Suurballe. Disjoint paths in a network. *Networks*, 4(2):125–145, 1974. 4

[39] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *CVPR*, 2014. 1

[40] D. Walden. The bellman-ford algorithm and "distributed bellman-ford". Technical report, MIT, 2003. 4

[41] X. Wang, E. Turetken, F. Fleuret, and P. Fua. Tracking interacting objects optimally using integer programming. In *ECCV*, 2014. 2

[42] X. Wang, M. Yang, S. Zhu, and Y. Lin. Regionlets for generic object detection. In *ICCV*, 2013. 6, 7

[43] B. Yang and R. Nevatia. An online learned crf model for multi-target tracking. In *CVPR*, 2012. 1, 2

[44] M. Yang, Y. Liu, L. Wen, Z. You, and S. Z. Li. A probabilistic framework for multitarget tracking with mutual occlusions. In *CVPR*, 2014. 2

[45] 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. 2

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