Supplementary Material for 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

In this supplementary material we specify further technical details and show additional results. We first provide links to the supplementary videos showing the performance of our optimal and approximate solvers on several challenging sequences from the KITTI tracking dataset. Secondly, we specify all necessary details of the association features we employ as pairwise constraints and illustrate the logistic functions we have learned. Furthermore, we give additional results in terms of runtime and provide an analysis of the sensitivity of the proposed model against variation of the parameters. We also show additional quantitative and qualitative results comparing the optimal batch solution to the proposed memory and computationally bounded online variant. We conclude with a review of related algorithms for finding successive shortest paths (SSP) in a network in order to help understanding our technical contributions.

1. Videos

We encourage the reviewers to have a look at the supplementary videos:

- Qualitative Results for Cars
- Additional Qualitative Results for Cars, Pedestrians, and Cyclists.

The videos may be started by clicking on the items above .

2. Association Features

This section explains the features we use for associating detections. We start with the two appearance-based features we use:

- **Color Histogram Similarity** s_c computes a measure of how well the histograms of both detections match:

$$s_c(H_1, H_2) = \frac{\sum_i (H_1(i) - \bar{H}_1)(H_2(i) - \bar{H}_2)}{\sqrt{\sum_i (H_1(i) - \bar{H}_1)^2 \sum_i (H_2(i) - \bar{H}_2)^2}} \tag{1}$$

  where

$$\bar{H}_k = \frac{1}{n_p} \sum_j H_k(j)$$

  and n_p is the total number of histogram bins. Note that a normalization is performed by subtracting the mean $\bar{H}_k$ in the numerator and computing the standard deviation σ_H in the denominator. When we use color images, we compute the sc separately for each channel and average their values. In particular, we use the Lab color space which was designed to approximate visual human perception and does not suffer as heavily as, e.g. the RGB color space from illumination changes.

- **Normalized Cross-Correlation:** We also employed the normalized cross-correlation s_x between the two bounding boxes. Changes in the scale of the object are considered by computing the correlation for a range of scales and taking the maximum value. To handle uncertainties in the bounding box position, the template is cropped to 80% of its original size.

Positional-based image cues are primarily suitable for comparing objects in consecutive frames. Bounding box overlap, size, and position in the image plane are subject to perspective changes for non-adjacent frames and moving objects, considerably changing for a longer time period. In particular, we use four positional-based cues defined as follows:

- **Bounding Box Overlap** s_o is a thresholdded intersection over union (IoU) between the bounding boxes of both objects

$$s_o(\mathbf{b}_1, \mathbf{b}_2) = \begin{cases} IoU & \text{if } IoU > 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$\text{with} \quad IoU = \frac{\mathbf{b}_1 \cap \mathbf{b}_2}{\mathbf{b}_1 \cup \mathbf{b}_2}$$

- **Bounding Box Size Similarity** s_s is the normalized sum of the absolute width and height difference

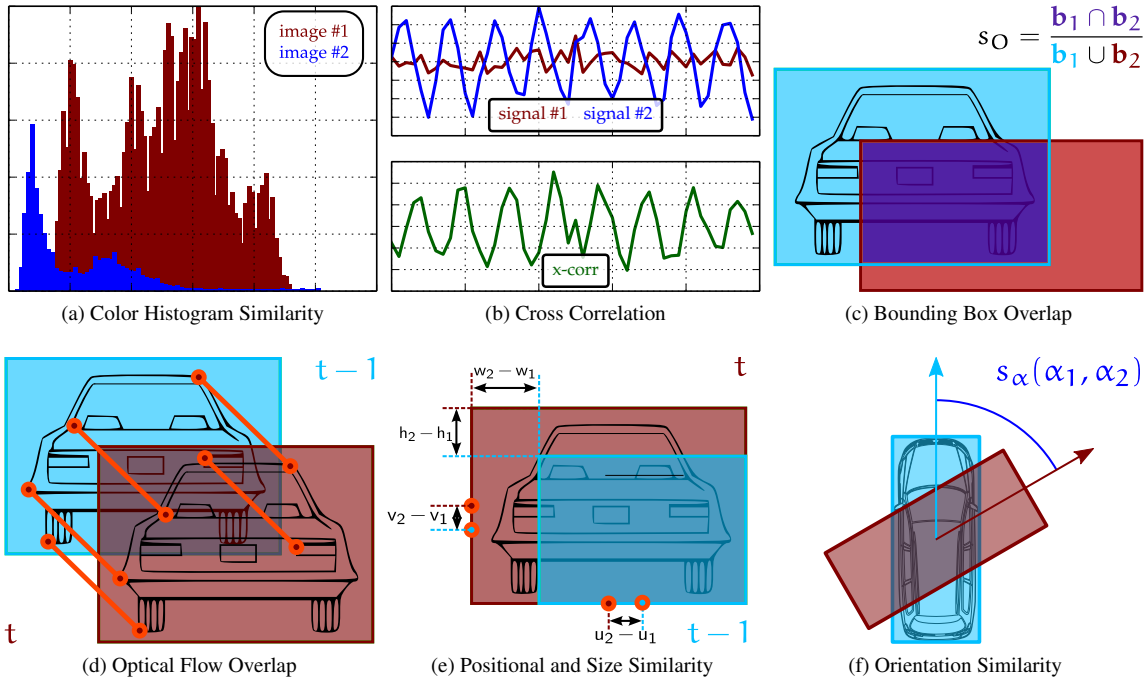
$$s_s(\mathbf{s}_1, \mathbf{s}_2) = 1 - \frac{1}{2}\left[\frac{|w_1 - w_2|}{\max(w_1, w_2)} + \frac{|h_1 - h_2|}{\max(h_1, h_2)}\right] \tag{3}$$

- **Location Similarity** s_p is the sum of absolute differences for the bounding box position normalized by the image dimensions.

$$s_p(\mathbf{u}_1, \mathbf{u}_2) = 1 - \frac{1}{2}\left[\frac{|\mathsf{u}_2 - \mathsf{u}_1|}{\mathsf{w}_{img}} + \frac{|\mathsf{v}_2 - \mathsf{v}_1|}{\mathsf{h}_{img}}\right] \tag{4}$$

- **Orientation Similarity** s_α is the normalized cosine similarity of the absolute angular difference

$$s_\alpha(\alpha_1, \alpha_2) = \frac{\cos(|\alpha_1 - \alpha_2|) + 1}{2} \tag{5}$$



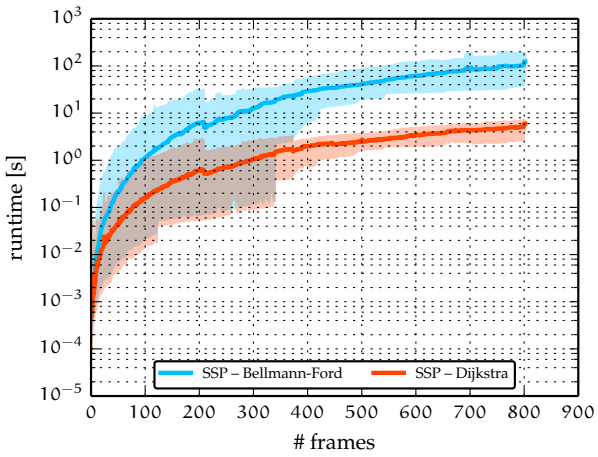
(a) Color Histogram Similarity (b) Cross Correlation (c) Bounding Box Overlap

(d) Optical Flow Overlap (e) Positional and Size Similarity (f) Orientation Similarity

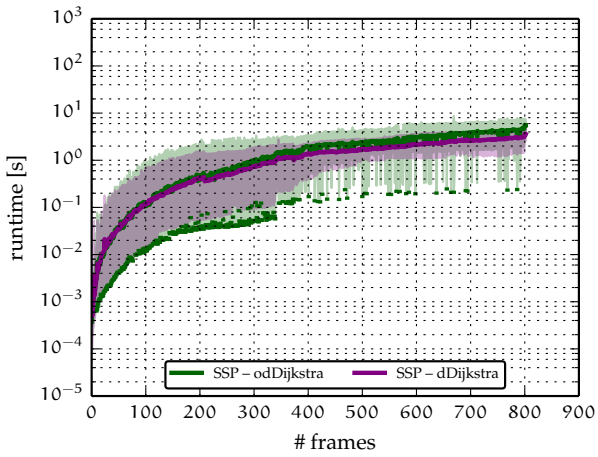
Figure 1: **Association Features.** Image-based ((a), (b)) and positional-based ((c), (d), (e), (f)) features used as input for association costs within the network flow formulation.

3. Runtime

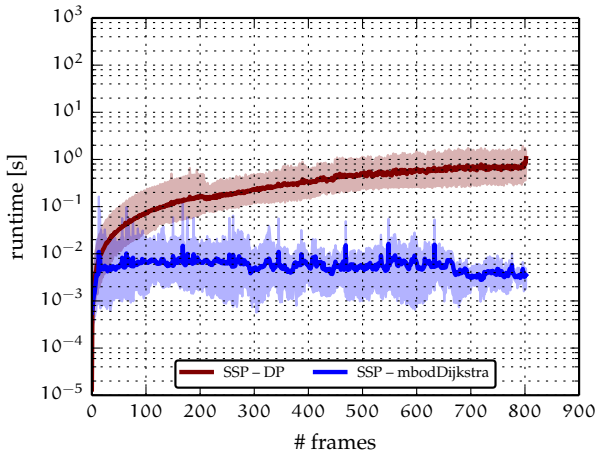
We used the training sequences of the KITTI tracking benchmark and show the mean run time and its variance to compute results up to a particular frame, where we average across sequences. In addition to Dijkstra's algorithm, we state the runtime for Bellman-Ford ("BF") as shortest path solver, which can handle the arbitrary costs of the original DAG and therefore avoids cost conversion. For Dijkstra and Bellman-Ford, we implemented the algorithms described in [6]. As shown in Fig. 2, our approach outperforms Dijkstra's algorithm by factor 2. For the online approach we noticed a slight overhead but speed-ups up to one order of magnitude were observed. Note that the worst case complexity cannot be reduced as in the worst case the whole predecessor map needs to be computed. However, this bound is loose, and the average case is much more favorable and the typical scenario when considering multi-target tracking.More importantly, our memory-bounded algorithm has a complexity independent of the sequence length and outperforms significantly in terms of runtime the algorithm of [12] (see Fig. 2(c)). Furthermore, detailed run times of the modules that composed the batch solvers are given in Fig. 3. Note that dDijkstra is called dSSP in the paper.



(a) Existing Baselines



(b) Proposed Optimal Solvers



(c) Approximate Solutions

Figure 2: **Comparing Run Time:** Batch solvers are shown as a function of the sequence length in the first two figures. As shown in the last figure, our approach outperforms significantly [12], named SSP-DP in terms of runtime. We use a window size of $\tau = 10$ for mbodSSP and evaluate all methods on a the original network without pruning any edges.

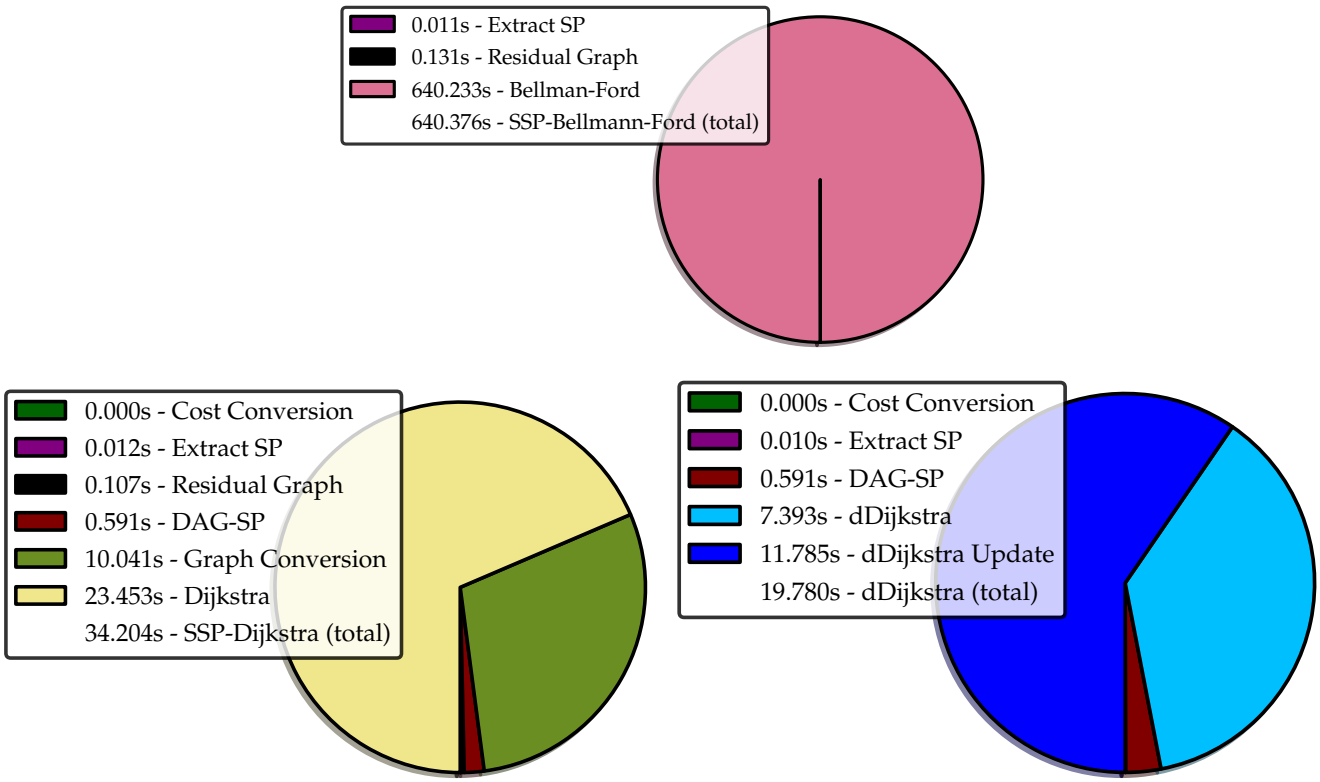


Figure 3: **Run Time Analysis of the Different Batch Solvers:** SSP with Bellman-Ford (top left), Dijkstra (top right), dDijkstra (dSSP in the paper) (bottom) as shortest path solver. The Bellman-Ford algorithm does not need to convert the graph and therefore spends all resources relaxing edges. Our dynamic algorithm (dDijkstra) decreases the run time necessary to find the shortest path by factor 3 when comparing to the standard Dijkstra algorithm. Note that there is also a slight overhead for creating the residual graph (Graph Conversion/Update).

4. Sensitivity to the Parameters

Fig. 4 depicts the logistic functions learned for the different features. As shown in Fig. 5 and Fig. 6 the performance does not depend too much on the specific value of the parameters. The learned values for computing the final edge costs are depicted in Table 1. Fig. 7 shows the performance of our memory bounded algorithm mbodSSP as a function of the window size τ (i.e., history length). Note that the curves are very flat, and good performance is achieved with a small window. The results for an infinite history length (i.e., batch method) are shown on the right hand side.

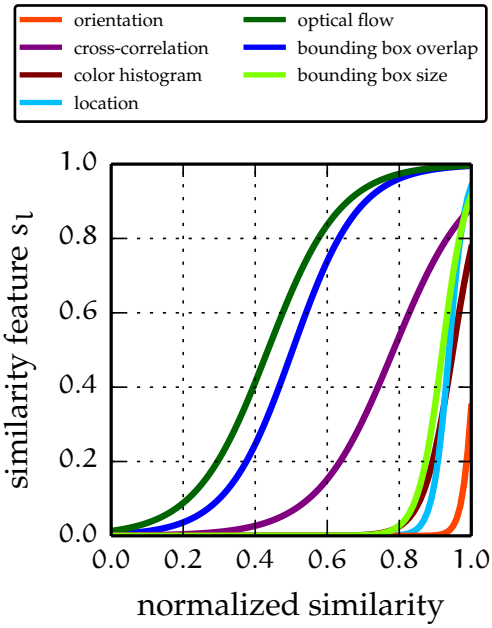
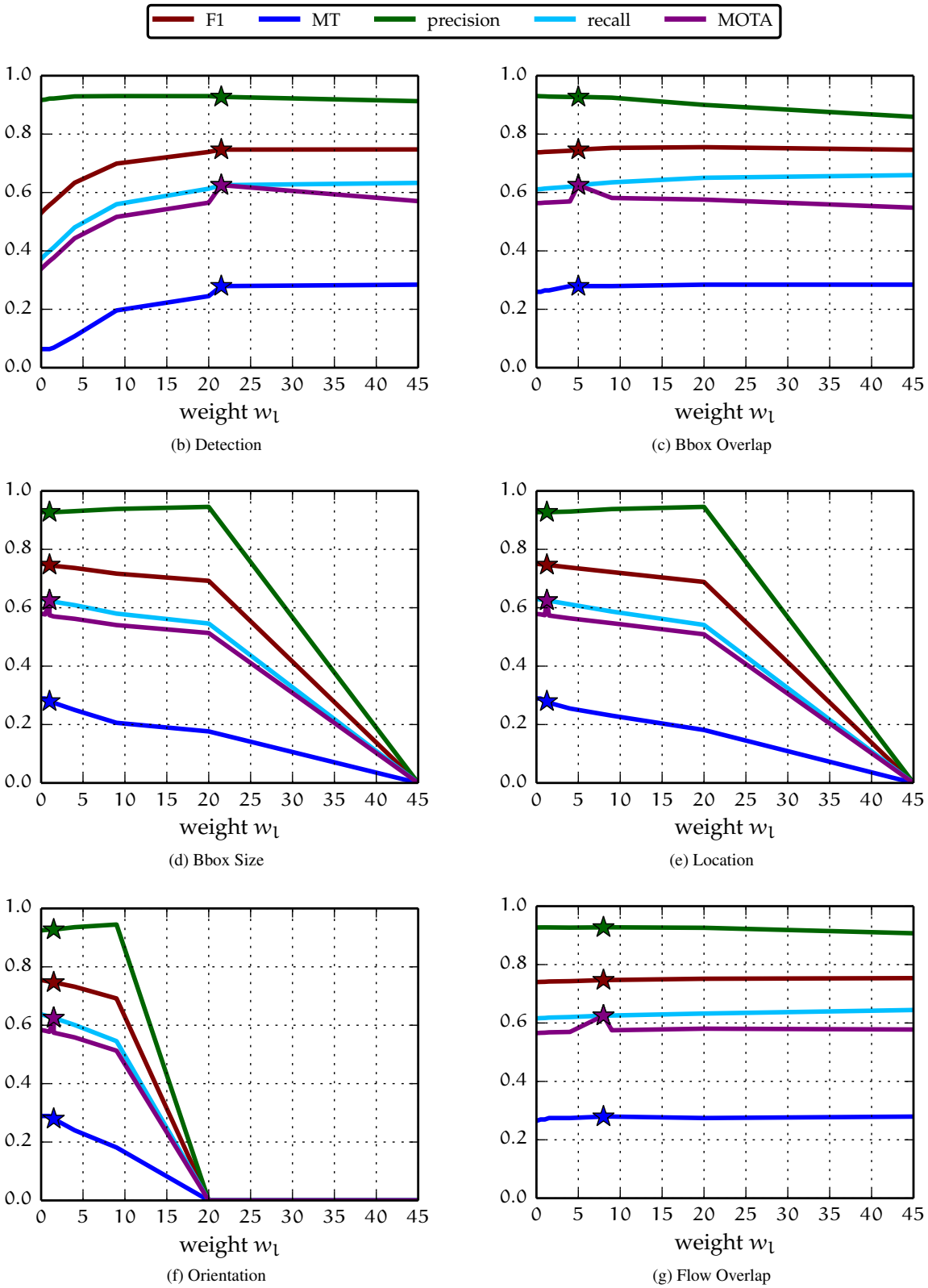


Figure 4: **Similarity Features.** Logistic functions learned that transform each raw feature l (*e.g.*, bounding box IOU) into an association score s_l (similarity). We learn all logistic functions via logistic regression on the KITTI training set using the ground truth associations.

FEATURE	en	ex	det	α	over	size	color	x-corr	loc	flow	β
w_l	5.0	5.0	21.5	1.5	5.0	1.0	9.25	6.5	1.25	8.0	8.4
o_l	0.0	0.0	-0.5	1.0	-0.75	0.0	-0.8	-0.01	0.625	-0.4	0.0

Table 1: **Model parameters.** This table shows the weight w_l and offset o_l of the logistic function for each feature l which we kept fixed during all our experiments. The robustness of each feature l against variation of its weight value w_l is shown in Fig. 5 and Fig. 6 for different values of w_l.



(b) Detection

(c) Bbox Overlap

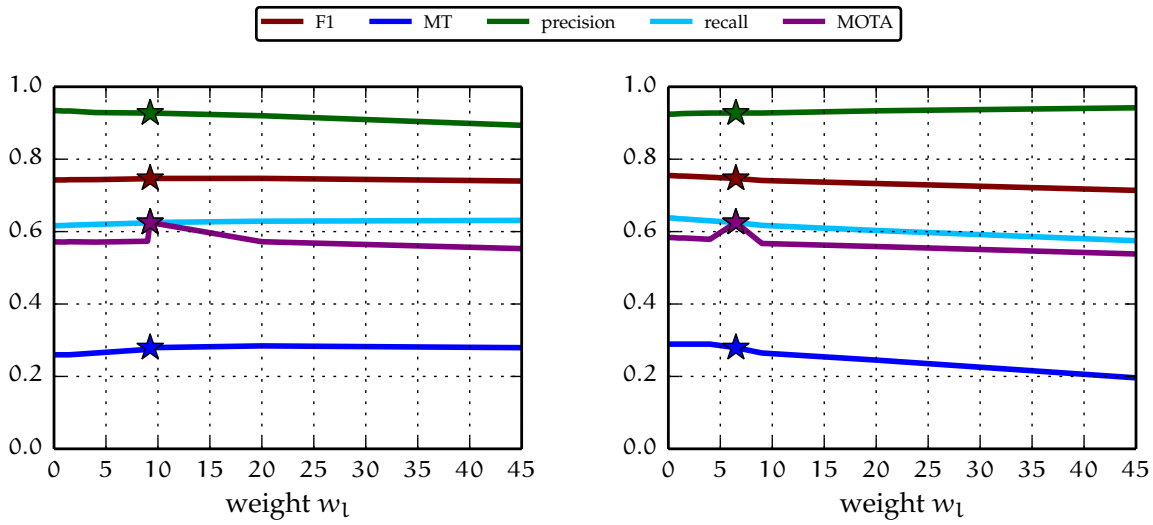
(d) Bbox Size

(e) Location

(f) Orientation

(g) Flow Overlap

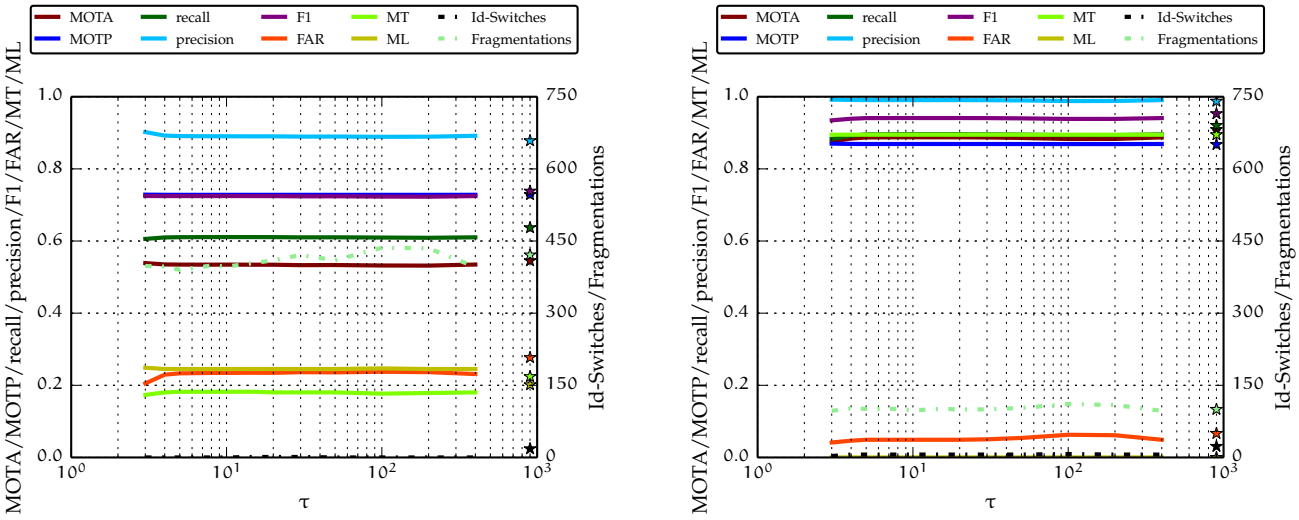
Figure 5: **Robustness of Detection and Positional Features against Variation of Parameters.** This figure shows the performance for different weight values on the KITTI training set using dSSP. While varying one parameter, all other parameters are kept fixed (c.f. Table 1).



(b) Color Histogram

(c) Cross-Correlation

Figure 6: **Robustness of Appearance-based Features against Variation of Parameters.** This figure shows the performance dSSP for different weight values on the KITTI training set. While varying one parameter, all other parameters are kept fixed (c.f. Table 1).



(a) KITTI (car)

(b) PETS (pedestrian)

Figure 7: **Performance for Different History Lengths.** We evaluated the performance of mbodSSP for different values of its history length τ on the whole KITTI training set and PETS S2.L1. Result for $\tau \to \infty$ are given on the very right of the plot.

5. Additional Quantitative Results

In this section we provide the full result tables with all metrics employed in the literature [5, 10] for both our methods and the baselines on KITTI (Table 2 + 3) as well as PETS 2009 (5) dataset. Note that our approach performs better than competing approaches in almost all metrics.

	HM	[3]	[11]	[12]	[8]	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
MODA	0.42	0.36	0.48	0.52	0.52	0.52	**0.54**	0.67	0.67
MODP	0.53	0.54	0.58	0.54	0.58	**0.59**	**0.59**	0.66	0.67
Recall	0.43	0.50	0.54	0.46	0.54	**0.56**	**0.58**	0.78	0.80
Prec.	**0.97**	0.77	0.90	0.96	0.95	0.93	0.94	0.88	0.87
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
PT	0.50	**0.56**	0.52	0.49	0.52	0.55	0.51	0.55	0.51
ML	0.42	0.34	0.34	0.39	0.35	**0.30**	**0.27**	0.10	0.090
TP	15293	17734	19057	16113	19308	**19909**	**20447**	28623	29714
FP	**535**	5161	2045	588	923	1588	1279	3765	4500
Misses	20121	17733	16401	19287	16139	**15506**	**15034**	8022	7279
IDS	12	223	125	2738	33	**0**	**7**	117	194
Frag.	578	624	**401**	3241	540	708	717	894	977

Table 2: Comparison of the proposed methods to four state of the art methods and a HM baseline implementation on KITTI (car) [9] using DPM detections. An asterisk in the method name indicates usage of Regionlet detections.

	[11]	mbodSSP	SSP
MOTA	**0.36**	0.13	0.16
MOTP	**0.75**	0.67	0.67
MODA	**0.37**	0.13	0.16
MODP	**0.76**	0.16	0.17
Recall	**0.47**	0.33	0.37
Precision	**0.82**	0.62	0.64
F1	**0.60**	0.43	0.46
FAR	0.56	**0.42**	0.43
MT	**0.080**	0.024	0.048
PT	0.39	0.44	**0.50**
ML	0.53	0.54	**0.45**
TP	**27568**	7641	8513
FP	6221	**4620**	4830
Misses	31185	15653	**14791**
IDS	221	**4**	111
Fragmentations	1011	**929**	1134

Table 3: Comparison of the proposed methods one state of the art method on KITTI (pedestrian) [9] using DPM detections.

	[2]	[3]	EKF [11]	[11]	mbodSSP	SSP
MOTA	0.81	**0.96**	0.68	0.91	0.89	0.91
MOTP	0.76	0.79	0.77	0.80	**0.87**	**0.87**
MODA	n/a	n/a	n/a	n/a	0.89	0.91
MODP	n/a	n/a	n/a	n/a	0.87	0.87
Recall	n/a	n/a	0.70	**0.92**	0.90	**0.92**
Prec.	n/a	n/a	0.98	0.98	**0.99**	**0.99**
F1	n/a	n/a	0.82	**0.95**	0.94	**0.95**
FAR	n/a	n/a	0.08	0.07	**0.057**	0.067
MT	0.83	**0.96**	0.39	0.91	0.89	0.89
PT	**0.17**	0.04	0.57	0.05	0.11	0.11
ML	**0.0**	**0.0**	0.04	0.04	**0.0**	**0.0**
TP	n/a	n/a	n/a	n/a	4153	4247
FP	n/a	n/a	65	59	**45**	53
Misses	n/a	n/a	1173	**302**	461	367
IDS	15	10	25	11	**7**	23
Frag.	21	8	30	**6**	100	100

Table 4: Comparison of the proposed method to three baselines on PETS 2009 [7].

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 5: Comparison for different values for the approximation parameter τ.

6. Additional Qualitative Results

In this section, we show additional qualitative results for several sequences from the KITTI tracking benchmark (Fig. 8 – 13). Additionally, we show how our method performs when tracking cars, pedestrians as well as cyclist in challenging crowded scenarios (Fig. 14+15). For further details, we encourage the reviewers to have a look at the supplementary videos:

- Qualitative Results for Cars
- Additional Qualitative Results for Cars, Pedestrians, and Cyclists.

The videos may be started by clicking on the items above or any of the respective images in the related figures.

All figures show results for the optimal solution (left) and our approximate memory-bounded approach (right). We start by discussing notably good results (Fig. 8 – 10) and review differences between the optimal and approximate solution as well as some failure cases (Fig. 11 – 13). All examples are discussed in the caption of the respective figure.

Figure 8: **Qualitative Results for the Optimal (left) and our Approximate Solution (right).** Far away objects are well tracked. Track 13/11 is followed for 3.7s and not confused with several parking cars.

Figure 9: **Qualitative Results for the Optimal (left) and our Approximate Solution (right).** Objects are tracked despite sharp turns. Significantly changing bounding box sizes and appearance do not lead to track interruption. (track 3/0)

Figure 10: **Qualitative Results for the Optimal (left) and our Approximate Solution (right).** Preceding objects in crowded scenes are well tracked. Track 3/1 is followed for 9.5s

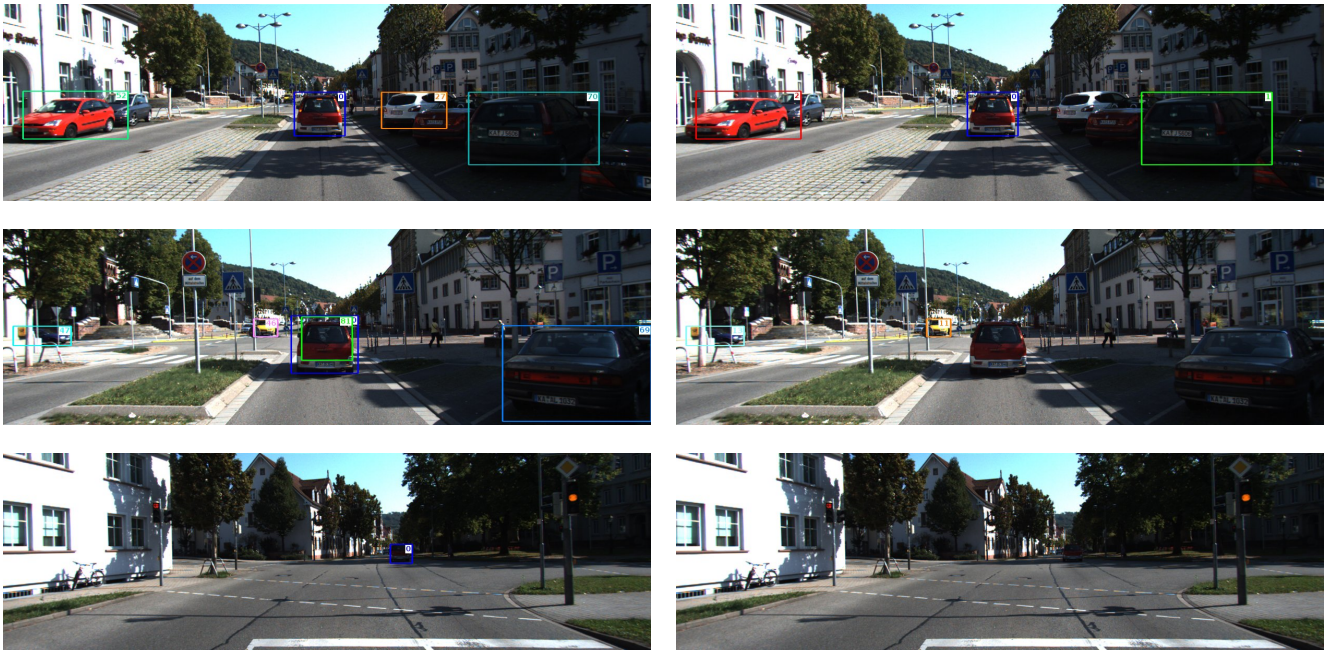


Figure 11: **Qualitative Results for the Optimal (left) and our Approximate Solution (right).** Changes in illumination and resulting ambiguous detections (middle) may lead the approximate solution to terminate a track due to missing future evidence. In contrast, the optimal solution tracks the preceding car for 37.2s.

Figure 12: **Qualitative Results for the Optimal (left) and our Approximate Solution (right).** Challenging intersection scenarios lead to qualitatively good results. The limited optimization window for the approximate solution may also prevent changes for parts of trajectory outside of this window. Due to this approximation, track 9/12 is not fragmented for mbodSSP.

Figure 13: **Qualitative Results for the Optimal (left) and our Approximate Solution (right).** Missing future evidence for the approximate solution may lead to tracks that start later when compared to the optimal solution (track 76/4). This happens primarily for detections with a low score for the early part of the trajectory.

Figure 14: **Qualitative Results for the Optimal (left) and our Approximate Solution (right).** For crowded inner-city scenes, objects of different classes (cars, pedestrians, cyclists) are tracked robustly.

Figure 15: **Qualitative Results for the Optimal (left) and our Approximate Solution (right).** For groups of interacting pedestrians, tracks are correct even for partly occluded objects. Very high occlusions for several frames may lead to interrupted tracks.

7. A review on Network Algorithms for Shortest Paths Computation

In this section, we discuss shortest path solvers for graphs with arbitrary costs and no negative cycles (Algo. 1), positive costs only (Algo. 2) as well as algorithms for finding successive shortest paths in both cases (SSP Algo. 3 for arbitrary costs, KSP Algo. 4 for positive costs only).

Graphs with Arbitrary Costs We start our discussion by describing the Bellman-Ford (BF) algorithm (Algo. 1), which is typically called at each SSP iteration for computing the shortest path on the residual graph $G_r(f)$. BF is based on the principle of *relaxation*, in which an upper bound of the correct distance to the source for each node 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 $|V|-1$ iterations, where $|V|$ denotes the number of vertices in the graph. In each of these repetitions, the number of vertices with correctly calculated distances grows, from which it follows that eventually all vertices will have their correct distances, even before the $|V|-1$'th repetition. In contrast to the originally proposed BF algorithm, current implementations consider this behavior by terminating the algorithm early when no changes between two iterations are detected anymore.

Algorithm 1: Bellman-Ford [6, p.651ff.]

Input: Graph G, Costs C, Source s
Output: Predecessor Map π, Distance Map d
// set $\pi(u) = \texttt{Unkown}, d(u) = \infty,\ u \in |V| \setminus s$
$\pi, d \leftarrow$ InitializeSingleSource(G,s)
for $i \leftarrow 1, \ldots, |V|-1$ **do**
  foreach *edge* $(u, v) \in G$ **do**
    $\pi(v) \leftarrow$ Relax($\pi(v)$, (u, v), $c(u, v)$)
// check for negative cycles
foreach *edge* $(u, v) \in G$ **do**
  if $d(v) > d(u) + c(u, v)$ **then**
    // a negative cycle was detected
    RaiseError("negative cycle")
return $\pi,\ d$

While the BF algorithm is able to compute a single trajectory with 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 Algo. 3) for a given example. Let us consider the *directed acyclic graph* (*DAG*) in Fig. 16(a) which specifies an instance of the network flow problem for four frames and three detections per frame. First, we relax all edges by traversing the graph from left-to-right, which is illustrated in Fig. 16 (b)+(c) for the first and second time step. This allows to recover the first optimal trajectory depicted in green in Fig. 16(d). We refer to this algorithm as *DAG-SP* as stated in [6, p. 655]. An alternative at this point is to remove all edges from this trajectory and run the algorithm again [12]. However, this greedy version of the algorithm does not guarantee optimality, as shown in Fig. 16(e) where some of the edges are absent in the graph. Instead, we introduce a residual graph $G_r^{(k)}$ (Fig. 16(f)) by inverting the edges of the previously found optimal trajectory. The resulting graph is, however, not a DAG anymore. A slight improvement of the approximate greedy solution can be achieved by considering backward-pointing edges once by traversing through the graph against the direction of the initial DAG [12]. In the following, we refer to this heuristic approximation as *dynamic programming* (DP). Instead, the SSP algorithm applies BF again on the residual graph (Fig. 16(g)) yielding the second shortest path and preserving optimality. The process finishes when a newly found shortest path has positive costs, *i.e.*, it can not further reduce the total cost. The final trajectories shown in Fig. 16(i) are recovered by extracting all backward edges from the most recent residual graph, starting the back-tracking procedure at the target node.

Graphs with Positive Costs After stating the SSP algorithm for graphs with arbitrary edge weights, we continue our exposition with the KSP algorithm [4, 13, 14] for a graph with positive costs only. Since the original tracking problem contains negative edge weights, an initial conversion of the graph is required. This can be done after initially running the DAG-SP algorithm for the topologically ordered graph, resulting in a predecessor map containing the shortest path to every

Algorithm 2: Dijkstra's Algorithm [6, p. 658ff.]

Input: Graph G, Source s, Costs c
Output: Predecessor Map π, Distance Map d
// set $\pi(u) = \texttt{Unkown}, d(u) = \infty,\ u \in |V| \setminus s$
$\pi, d \leftarrow$ InitializeSingleSource(G,s)
$S \leftarrow \emptyset$ // invariant nodes
$Q \leftarrow$ edges(G) // min-priority queue
while $\neg$ *empty(Q)* **do**
  $u \leftarrow$ NodeWithMinDistance(Q)
  $S \leftarrow S \cup \{u\}$
  foreach *node* $v \in$ *Neighbors(G,u)* **do**
    // Check if u is a better predecessor for v
    $\pi(v) \leftarrow$ Relax(u,v,c)
    if $d(v) > d(u) + c(u, v)$ **then**
      $d(v) \leftarrow d(u) + c(u, v)$
return π,d

Algorithm 3: Successive shortest-path (SSP) [1]

Input: Detections $\mathcal{X} = \{\mathbf{x_i}\}$, Source s, Target t
Output: Set of trajectory hypotheses $\mathcal{T} = \{T_k\}$
// construct graph from observations
$G(V, E, C, f) \leftarrow \mathcal{X}$
$f(G) \leftarrow 0$ // initialize flow to 0
$G_r(f) \leftarrow G(f)$ // initialize $G_r(f)$ as a DAG
// find k-th shortest path
while $C(\gamma_k) < 0$ **do**
  // using any SP solver
  $\gamma_k \leftarrow$ FindShortestPath($G_r(f), s, t$)
  // Revert edges for $G_r(f)$ along γ_k
  $G_r(f) \leftarrow$ RevertEdges($G_r(f)$,γ_k)
return $\mathcal{T}$

node of the graph. Exploiting this predecessor map, arbitrary edge weights $C_{i,j} \in \mathbb{R}$ can be converted to weights $C'_{i,j} \in \mathbb{R}_0^+$ by

$$C'_{i,j} = C_{i,j} + d(i) - d(j) \quad \forall e_{i,j} \in E \tag{6}$$

with $d(i)$ the distance on the shortest path from the source to node i. This results in a graph with only positive costs (Fig. 17). Edges on a shortest path of this converted graph have a weight $C'_{i,j} = 0$ describing an arborescence Γ [15, p. 126ff.]. An example of this conversion is depicted in Fig. 18. Reducing costs in this way must be performed for every residual graph. However, the runtime for this step is negligible compared to computing the shortest path.

We continue by reviewing KSP and start with Dijkstra's algorithm for finding shortest paths as stated in Algo. 2. In each KSP iteration, Dijkstra is initially called on the residual graph $G_r^{(k)}$. As BF, Dijkstra is based on the principle of relaxation. Since the minimum cost of an edge is guaranteed to be $C'_{i,j} = 0$, the algorithm exploits this property as a lower bound for the best shortest path. Starting at the source node, a priority queue is maintained and in every iteration all edges from the most promising node u to its successors v are relaxed. For every node u taken from the queue, the final shortest path was determined. Consequently, while establishing a particular s-t path, the algorithm may terminate early after taking the target node from the queue. However, within the KSP algorithm the shortest path to every node must be computed as mentioned above, whereas an early termination may leave the predecessor map in an incomplete state.

After detecting a shortest path on $G_r^{(k)}$, an iteration of the KSP algorithm is continued by converting again the edge costs

Algorithm 4: K-Shortest Paths (KSP) [4, 13]

Input: Detections $\mathcal{X} = \{\mathbf{x_i}\}$, Source s, Target t
Output: Set of trajectory hypotheses $\mathcal{T} = \{T_k\}$

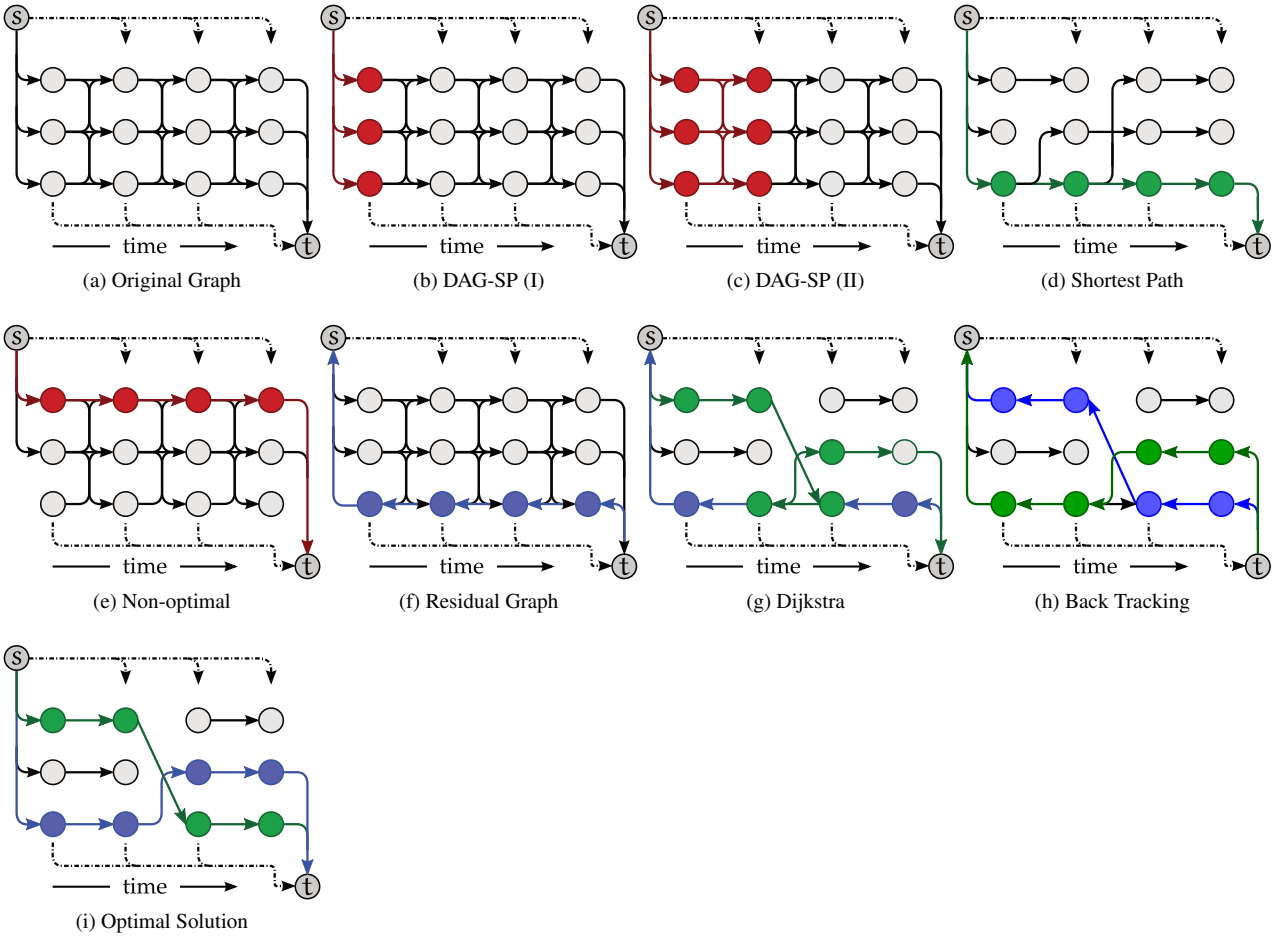
$G(V, E, C, f) \leftarrow$ ConstructGraph$(\mathcal{X}, s, t)$ // DAG
$f(G) \leftarrow 0$ // initialize flow to 0
$G_r^{(0)}(f) \leftarrow G(f)$
$\pi^{(0)} \leftarrow$ DAG-SP$(G_r^{(0)}(f))$ // shortest path in DAG
// encode arborescence in residual graph
$G_r^{(0)}(f) \leftarrow$ ConvertEdgeCost$(G_r^{(0)}(f), \pi^{(0)})$
// Revert edges along shortest path
$G_r^{(1)}(f) \leftarrow$ ComputeResidualGraph$(G_r^{(0)}(f), \pi^{(0)})$
// find k-th shortest path
$k \leftarrow 0$
while 1 **do**
  $k \leftarrow k + 1$
  $\gamma^{(k)}, \pi^{(k)} \leftarrow$ Dijkstra$(G_r^{(k)}(f))$ // shortest path
  $G_r^{(k)}(f) \leftarrow$ ConvertEdgeCost$(G_r^{(k)}(f), \pi^{(k)})$
  $G_r^{(k+1)}(f) \leftarrow$ ComputeResidualGraph$(G_r^{(k)}(f), \gamma^{(k)})$
  // evaluate converted costs
  if $\sum_{i=1}^{k} \text{cost}(\gamma^{(i)}) > |\text{cost}(\gamma^{(0)})|$ **then**
    break
return $\mathcal{T}$

of $G_r^{(k)}$ according to Eq. 6 resulting in a graph containing the arborescence

$$\Gamma = G_r^{(k)} \setminus \{E \mid C'_{i,j} \neq 0\}. \tag{7}$$

Finally, the subsequent residual graph $G_r^{(k+1)}$ is computed according to [13] by reverting the direction of edges along the shortest path $\gamma^{(k)}$. While for the original graph the SSP algorithm terminated after finding a path with positive costs, the KSP algorithm for the converted problem finishes if the sum of the total costs $\sum_{i=1}^{k} \text{cost}(\gamma^{(i)})$ are higher compared to the first iteration $|\text{cost}(\gamma^{(0)}|$. The KSP algorithm is summarized as Algo. 4.

Comparing the time complexity of both SSP and KSP, the differences come from the time complexities of the shortest path solver, which must be repeated $n_K + 1$ times for n_K objects present in the scene. The total complexity of the BF algorithm is $\mathcal{O}(|V||E|)$, where $|V|$ is the number of vertices and $|E|$ is the number of edges. Basically, BF relaxes all edges as many times as vertices exist in the graph to guarantee the shortest path to every node. This is typically too expensive for real-world applications. In contrast, the lower bound and the priority queue used in Dijkstra's algorithm reduce the worst-case complexity, depending on the complexity for queue operations. Using an efficient Fibonacci heap for this task results in a time complexity of $\mathcal{O}(|E| + |V| \log(|V|))$ resulting in KSP ($\mathcal{O}(n_K(|E||V| \log(|V|))$) being more efficient than SSP ($\mathcal{O}(n_K|V||E|)$). The non-optimal DP algorithm [12] still saves $\log(|V|)$ operations over using KSP.



(a) Original Graph (b) DAG-SP (I) (c) DAG-SP (II) (d) Shortest Path

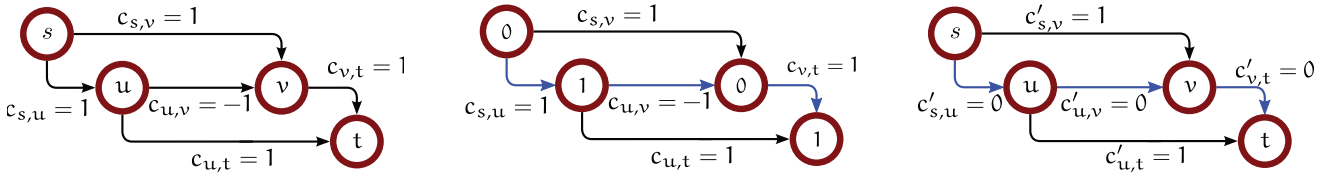
(e) Non-optimal (f) Residual Graph (g) Dijkstra (h) Back Tracking

(i) Optimal Solution

Figure 16: **Illustration of the SSP Algorithm.** The original problem is depicted in (a). Dynamic programming 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 the algorithm violates optimality as shown in (e). Instead, a SSP strategy on the residual graph (f) allows for finding the optimal solution (g+h). Backward pointing edges in the residual graph (h) encode trajectories for the original problem (i).

(a) Original Costs (b) Converted Costs

Figure 17: **Cost Conversion.** For an edge between the nodes u and v, costs $C_{u,v} \in \mathbb{R}$ can be converted into $C'_{u,v} = C_{u,v} + d(u) - d(v) \in \mathbb{R}_0^+$. Therefore, the distance on the shortest path $\mathrm{d}(\cdot)$ from the source to both nodes u and v must known.



(a) Original Graph (b) Shortest Path (c) Converted Graph

Figure 18: **Cost Conversion Example.** For the graph in (a) with negative costs, the shortest s-t path is computed (blue edges in (b)). Numbers in the nodes state the distance on the shortest path for this particular node. The graph is converted into a graph without negative costs according to Eq. 6. The resulting graph in (c) contains the arborescence Γ in blue which is identical with the shortest s-t path in this case.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice Hall, 1993. 15

[2] A. Andriyenko and K. Schindler. Multi-target tracking by continuous energy minimization. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2011. 9

[3] A. Andriyenko, K. Schindler, and S. Roth. Discrete-continuous optimization for multi-target tracking. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012. 8, 9

[4] J. Berclaz, F. Fleuret, E. Turetken, and P. Fua. Multiple object tracking using k-shortest paths optimization. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 33(9):1806–1819, 2011. 14, 16

[5] K. Bernardin and R. Stiefelhagen. Evaluating multiple object tracking performance: The clear mot metrics. *Journal on Image and Video Processing (JIVP)*, 1:1–10, 2008. 8

[6] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001. 3, 14, 15

[7] A. Ellis and J. M. Ferryman. PETS 2010 and PETS 2009 evaluation of results using individual ground truthed single views. 2010. 9

[8] A. Geiger, M. Lauer, C. Wojek, C. Stiller, and R. Urtasun. 3D traffic scene understanding from movable platforms. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 36(5):1012–1025, 2014. 8

[9] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012. 8

[10] Y. Li, C. Huang, and R. Nevatia. Learning to associate: HybridBoosted multi-target tracker for crowded scene. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2009. 8

[11] A. Milan, S. Roth, and K. Schindler. Continuous energy minimization for multitarget tracking. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 36(1):58–72, 2014. 8, 9

[12] H. Pirsiavash, D. Ramanan, and C. C. Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2011. 3, 8, 14, 16

[13] J. W. Suurballe. Disjoint paths in a network. *Networks*, 4(2):125–145, 1974. 14, 16

[14] J. W. Suurballe and R. E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14(2):325–336, 1984. 14

[15] W. Tutte. *Graph Theory*. Cambridge Mathematical Library. Cambridge University Press, 2001. 15